Generische Interaktionsmuster für aufgabenorientierte Dialogsysteme

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
der Fakultät für Informatik
an der Universität Karlsruhe (Technische Hochschule)
vorgelegte

Dissertation

von

Matthias Denecke

aus Hameln

Tag der mündlichen Prüfung: 23.05.2002
Betreuer: Prof.Dr.Alexander Waibel, Universität Karlsruhe
Korreferent: Prof.Dr.Christian Rohrer, Universität Stuttgart

Zusammenfassung Der Entwurf, die Implementierung und das Testen von natürlichsprachlichen Dialogsystemen ist eine zeitintensive Aufgabe. Erste Implementierungen von Dialogsystemen im Rahmen dieser Arbeit führten zu der Beobachtung, daß die Parameter, welche ein Dialogsystem definieren, grob in zwei Klassen unterteilt werden können. Die erste Klasse definiert aufgabenabhängiges Wissen, während die zweite Klasse interaktionsspezifisches Wissen definiert. Beide Klassen lassen sich weiter unterteilen. Die erste Klasse besteht aus folgenden Wissenquellen: (i) einer Ontologie, welche die Konzepte definiert, die das Dialogsystem 'versteht', (ii) Dialogzielbeschreibungen, welche die Aufgaben beschreiben, die das Dialogsystem ausführen kann, (iii) Satzanalyse-Grammatiken, welche die Spracherkennerausgabe mithilfe struktureller Beschreibungen in eine semantische Repräsentation in einer typisierten Merkmalslogik umwandeln, (iv) Datenbankbeschreibungen, welche die Referenz von Nominalphrasen auflösen, und (v) Generierungsgrammatiken, welche Konzepte der typisierten Merkmalslogik zurück in natürlichsprachliche Ausdrücke umformt. Die aufgabenabhängigen Wissensquellen, zusammen mit auf ihnen operierenden Algorithmen, bilden die erste Schicht einer dreischichtigen Dialogsystemarchitektur. Die in dieser Schicht realisierten Dienste werden *Basis-Dialogdienste* genannt.

Die zweite Schicht des Dialogsystems besteht aus den interaktionsspezifischen Wissensquellen. Hier wird die Art und Weise der Interaktion zwischen System und Benutzer implementiert. Die zweite Schicht setzt sich aus einem Katalog von *Interaktionsmustern* zusammen. Ein Interaktionsmuster beschreibt, wie Information im Diskurs angepaßt wird, legt aber nicht fest, wie diese Information vom Benutzer übermittelt wird. Zum Beispiel kann eine Menge von Optionen dem Benutzer durch nur eine Frage präsentiert werden *Wünschen Sie a,b,c oder* d*?* oder durch eine Reihe von Fragen der Art *Wünschen Sie a? Bitte sagen Sie ja oder nein.* Beide Strategien habe Vor- und Nachteile und können Defizite anderer Komponenten, z.Bsp. des Spracherkenners, teilweise ausgleichen. Das diese Strategien implementierende Interaktionsmuster ist jedoch das gleiche, nämlich das *Frage-Interaktionsmuster.* Neben dem frage-Interaktionsmuster, welches die Spezifizität der Information im Diskurs erhöht, gibt es das *Korrektur-Interaktionsmuster*, welches Information aus dem Diskurs entfernt, das *Anpassungs-Interaktionsmuster*, welches Informationen aus dem Diskurs ersetzt und das *Zustands-Interaktionsmuster*, welches die Information im Diskurs unverändert läßt, aber für die Meta-Dialogsteuerung zuständig ist. Ausgewählt werden die Interaktionsmuster von der dritten Schicht, der *Dialog-Steuerungs-Schicht.*

Die in dieser Arbeit untersuchte Fragestellung ist, wie die Entwurfszeit eines Dialogsystems durch die Einführung einer auf generellen Prinzipien basierenden modularen Architektur verkürzt werden kann. Der Hauptbeitrag dieser Arbeit ist die Entwicklung und experimentelle Verifizierung eines Systems von Interaktionsdiensten, welche Benutzeraktionen mit dem natürlichsprachlichen System zielsprach- und aufgabenunabhängig beschreiben und implementieren. Da Interaktionsdienste und Dialogstrategien durch die Basis-Dialogdienste formuliert werden können, ist die Unabhängigkeit von Zielsprache und Anwendungsdomäne gegeben. Deswegen kann eine Implementierung des Interaktionsdienstsystems als Ausgangspunkt für ein natürlichsprachliches Dialogsystem dienen. Zusätzlich können die aufgabenabhängigen Wissensquellen durch die Verwendung von Techniken, wie sie aus objektorientierten Programmiersprachen bekannt sind, modularisiert werden. Dadurch kann eine weitere Verkürzung der Entwicklungszeit erzielt werden. Es konnte experimentell nachgewiesen werden, daß in sieben Fällen mit Testpersonen, die mit der Systemarchitektur nicht vertraut waren, prototypische Systeme in sehr kurzer Zeit (etwa acht bis zwölf Stunden) entwickeln konnten. Diese Systeme können dann zur Datensammlung

eingesetzt werden. Die Abdeckung des Dialogsystems kann dann durch iterative Verfeinerung der beteiligten Wissensquellen verbessert werden.

Danksagung

Zunächst möchte ich meinem Doktorvater Prof. Dr. Alex Waibel herzlich danken. Er hat es mir ermöglicht, die Arbeit in einer erstklassigen Forschungsumgebung an der Carnegie Mellon University und der Universität Karlsruhe durchzuführen. Im Laufe der Jahre konnte ich das System an vielen interessanten Projekten ausprobieren und verbessern. Mein Dank gilt meinem Korreferenten Prof. Dr. Christian Rohrer. Er hat dazu beigetragen, daß die linguistischen Aspekte der Arbeit nicht vernachlässigt wurden.

I would also like to thank my colleagues at the Interactive Systems Laboratories at Carnegie Mellon University. I would like to thank Jie Yang for many helpful discussions and suggestions over the years. I learned a lot from you, Jie. When I first got to Pittsburgh, it was Minh Tue Vo who helped me getting started and getting involved with interactive map programs. Your help was very much appreciated, Minh. I would like to thank Laura Mayfield Tomokiyo, whose uncompromising love for the English language has improved the quality of many papers of mine she proofread. I enjoyed many insightful discussions with Alex Rudnicky that helped me focus my work. Klaus Ries and I met several times to discuss the progress (or lack thereof) of our respective theses; these meetings helped me stay on course. Also, I would like to thank the participants in the user study, Michael Bett, Andreas Hanemann, Ajay Juneja, Rob Malkin, Kay Petersen and Dorcas Wallace. Many thanks to my friends and colleagues at Carnegie Mellon University, Susanne Burger, Victoria MacLaren, Robert Malkin, Ariadna Font Llitjos, Dan Bohus and Bernhard Suhm. I would like to thank Marsal Gavalda with whom I shared an office at Interactive Systems, Inc. Our conversations were the highlight of that time. Do you still have that green ball? I would also like to thank Michael Bett for generally making things run smoothly and taking care of many things that often go unnoticed.

Ein besonderer Dank gilt meinen Karlsruher Kollegen, die mir während meiner vielen Besuchen in Karlsruhe immer Unterkunft, eine angenehme Arbeitsathmosphäre und immer viel abendliches Beiprogramm geboten haben. Besonderer Dank geht an Hagen Soltau, Florian Metze, Rainer Stiefelhagen und Martin Westphal für Unterbringung. Dank geht auch an Petra Gieselmann, Hartwig Holzapfel und Elin Topp für das tapfere Benutzen von ständig noch nicht ganz fertigen Versionen des Dialogsystems. Ein besonderer Dank geht an Silke Dannenmaier für die organisatorische Hilfe vor, während und nach der Prüfung und viele kleine Hilfen im Laufe der Jahre, die doch so wichtig sind.

A particular thank you goes to my wife Yu for never ending love and support, especially during the difficult times during the end of the thesis.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Maschinelle Sprachverarbeitung ist ein vergleichsweise altes Forschungsgebiet innerhalb der Informatik. Nur sechs Jahre, nachdem Konrad Zuse in 1943 den ersten Computer vorstellte, berichtete die *New York Times* vom 31. Mai 1949 von dem ersten Programm zur maschinellen Übersetzung. Aber erst heutzutage sind Sprachverarbeitungsprogramme in der Lage, die auf der Titelseite der *New York Times* gemachten Versprechen einzulösen – und das auch nur annähernd. Es scheint im Rückblick, daß die Schwierigkeiten im Gebiet der Künstlichen Intelligenz und der maschinellen Sprachverarbeitung im besonderen unterschätzt worden sind.

Was macht maschinelle Sprachverarbeitung zu einem schwierigen Problem? Da der Entwurf von Sprachverarbeitungsprogrammen eine interdisziplinäre Aufgabe ist, kommen mehrere Gründe zum Tragen. Zunächst ist die menschliche Sprache selbst von komplexer Struktur und Forschungsgegenstand für sich selbst. Dazu kommt, daß Menschen Zugriff über gewaltige Mengen von Weltwissen zur Verarbeitung von Sprache haben, ein Wissen, das Computern – trotz verschiedener Bemühungen wie zum Beispiel das Wissensrepräsentationssystem *Cyc* [Lenat, 1995] – nicht zur Verfügung steht. Aber selbst wenn man die Problemstellung der maschinellen Sprachverarbeitung auf eine begrenzte Domäne einschränkt, bleibt es ein schwieriges Problem. Das Erstellen und Testen von Resourcen, die für die Sprachverarbeitung notwendig sind, ist zeitaufwendig und mühsam.

Diese Arbeit schlägt eine Lösung für das letztgenannte Problem im Gebiet der natürlichsprachlichen Dialogsysteme vor. Wie können Dialogsysteme schnell entworfen werden, ohne daß Abstriche bei der Repräsentation oder Verarbeitung von komplexen Dialogen gemacht werden müssen? Wie diese Arbeit zeigt, kann dieses Ziel durch Techniken erreicht werden, die mit Erfolg im *Software Engineering* und *Knowledge Engineering* seit Jahren eingesetzt werden. Hierzu werden die Aspekte des Systems verkapselt, die sich beim Übergang von einer Domäne auf eine andere verändern. Aspekte des Systems, die konstant bleiben, werden aufgedeckt. Die im Rahmen dieser Arbeit entdeckten Invarianten, genannt *Interaktionsmuster*, sind Grundbausteine für natürlichsprachliche Dialogsysteme, die zielsprach- und domänenunabhängig die Sprachverarbeitungsfähigkeiten von aufgabenorientierten Dialogsystemen beschreiben.

1.1 Die Idee hinter Interaktionsmustern

Der zunehmenden Komplexität von Alltagsgegenständen Rechnung tragend, finden sich Computer in immer mehr Geräten des alltäglichen Lebens. Mit der Allgegenwart des Computers kommt das Problem der einfachen und einfach zu erlernenden Interaktion. Gesprochene Sprache als natürliches Kommunikationsmedium begründet das Interesse an natürlichsprachlichen Dialogsystemen. Ein natürlichsprachliches Dialogsystem, wie es im Rahmen dieser Arbeit verstanden wird, ist ein System, welches die Interaktion mit einer Maschine

1. durch Dialog
2. unter Benutzung natürlicher Alltagssprache
3. in einem Diskurs, der sich möglicherweise über mehr als einen Turn erstreckt
4. mit dem Ziel, der Maschine den vom Anwender gewünschten Dienst zu übermitteln und ausführen zu lassen, solange der Dienst Teil der vorgegebenen Domäne ist,

ermöglicht.

Aus Anwendersicht hat das Konzept des Sprachverstehens vielerlei Vorteile. Die Interaktion ist idealerweise natürlich, das Erlernen von Eingabesequenzen oder der Bedeutung von Knöpfen ist unnötig.

Aus technischer Sicht ist die Lage der Dinge weniger perfekt. Spracherkennung, trotz signifikanter Fortschritte in den letzten Jahren, ist immer noch mit Fehlern behaftet. Sprachverarbeitungssysteme sind sich immer noch vieler Subtilitäten der natürlichen Sprache nicht bewußt. Dazu kommt, daß die kombinatorische Komplexität von lexikalischen Einträgen das Formen von Ausdrücken erlaubt, die nicht nur vom Systemdesigner vorhergesehen, sondern auch von der Anwendung "verstanden", das heißt, in die richtigen Dienstanforderungen, umgesetzt werden müssen. Als Konsequenz davon kann es vorkommen, daß Computer einen gegebenen Satz zwar richtig verstehen, aber mit einer leichten Umformulierung des Satzes nicht zurechtkommen. Zusammengefaßt sind die unterschiedlichen Kompetenzen von Mensch und Computer in der Sprachverarbeitung ein Problem.

Als Konsequenz daraus müssen Dialogsysteme so entworfen werden, daß auch fehlerbehaftete Eingabe einen erfolgreichen Dialog ermöglicht. Dazu werden verschiedene Arten von Konsistenz- und Konfidenzüberprüfungen der Eingabe vorgenommen, und gegebenenfalls Teile der Eingabe ignoriert und dann durch interaktiven Dialog erfragt. Hierbei kann das System den Dialog steuern, um so die Wahrscheinlichkeit von Miskommunikation zu reduzieren. Es ist offensichtlich, daß der Entwurf, die Implementierung und das Testen von solchen Algorithmen zeit- und kostenintensiv ist.

Interessanterweise können aber die Algorithmen zur Dialogsteuerung zielsprach- und domänenunabhängig entworfen werden. Wenn beispielsweise in einem Dialog zwischen einem Anwender und einem Flugauskunftssystem die Rede von einem Ticket ist, muß das System erfragen, ob der Anwender das Ticket erstehen möchte oder in seiner Wunschliste speichern möchte. In einem Dialog zwischen einem Anwender und einem sprachverstehenden Wohnzimmer muß in einer ähnlichen Situation das System erfragen, ob eine Lampe ein- oder

ausgeschaltet werden soll. Ein einfacher Algorithmus, der diese Information erfragt, ist in Abbildung 1.1 gegeben.

Precondition: Object o under discussion is known
Action a to be applied to o is unknown
Action : infer applicable actions $a_1, \ldots, a_n$ from domain model
while $(n > 1)$ **begin**
prompt user for more information on the intended
incorporate prompted information
infer applicable actions $a_1, \ldots, a_n$ from domain model
end
Postcondition: Object o under discussion is known
Action a to be applied to o is known

Abbildung 1.1. Skizze eines vereinfachten Interaktionsmusters

Der Algorithmus erfragt, möglicherweise in mehr als einem Schritt, Information und fügt sie dem Diskurs hinzu, bis die bekannte Information eine gewisse Bedingung erfüllt (die Operation, die auf o angewandt werden soll, ist bekannt). Dieser Algorithmus kann gleichermaßen im Flugauskunftssystem und im interaktiven Wohnzimmer eingesetzt werden, solange die Bestandteile des Algorithmus allgemein genug formuliert sind.

Es soll jedoch bemerkt werden, daß der Algorithmus nicht alle zur Dialogausführung benötigten Spezifikationen enthält. Insbesondere werden keinerlei Angaben gemacht, *wie* zusätzliche Information akquiriert werden soll; es wird lediglich gefordert, *welche* Information erfragt werden soll. Im obigen Beispiel könnte der Algorithmus eine Frage der Form stellen: *"Möchten Sie die Flugverbindung ausdrucken oder möchten Sie das Ticket kaufen?"*. Alternativ dazu könnte die Frage gestellt werden: *"Möchten Sie die Flugverbindung ausdrucken? (Bitte sagen Sie ja oder nein)"*; und, wenn der Anwender mit *'nein'* antwortet, könnte eine Folgefrage erfragen, ob der Benutzer das Ticket kaufen möchte und so weiter. Mit anderen Worten, ein Interaktionsmuster spezifiziert, welche Information akquiriert werden soll, aber nicht wie dies geschieht. Damit ist eine Trennung von Interaktionsmustern und Dialogstrategie gegeben. Interaktionsmuster können damit unterschiedliche Dialogstrategien implementieren. Darüber hinaus kann es während desselben Dialogs unterschiedliche Instantiierungen desselben Interaktionsmusters geben. Entscheidungen über die Auswahl und Instantiierung eines Interaktionsmusters werden in einer übrgeordneten Komponente getroffen. Zu der Entscheidung über die Auswahl können Kriterien des bisher abgelaufenen Dialogs herangezogen werden. Zu den Kriterien gehören Parameter wie beispielsweise Konfidenzmaße des Spracherkenners, die Anzahl gestellter Fragen oder die Anzahl von Objekten, die durch einen Datenbankzugriff gefunden wurden.

1.2 Aussage der Arbeit und Beiträge

Im folgenden werden die in dieser Arbeit überprüften Thesen aufgestellt, die Forschungsmethodologie beschrieben und die Beiträge der Arbeit zusammengefaßt.

1.2.1 These

Es ist die Aussage dieser Dissertation, daß

(i) das dialogtechnische Verhalten von aufgabenorientierten Dialogsystemen durch die Angabe einer kleinen Menge von generischen Interaktionsmustern beschrieben werden kann;
(ii) robuste Implementierungen von natürlichsprachlichen Dialogsystemen in einer gegebenen Zielsprache und Domäne durch die Ergänzung von generischen Interaktionsmuster durch sprach- und domänenspezifische Wissensquellen erfolgen kann; und
(iii) dieser Ansatz gleichzeitig dazu geeignet ist, ein Konzept für ein System schnell zu entwickeln *Rapid Prototyping* wie auch ein schnell entwickeltes System zu einem robusten, voll funktionierenden System auszubauen.

1.2.2 Methodologie, Systementwurf und Implementierung

Der verfolgte Ansatz zur Überprüfung der oben aufgestellten Thesen war, die benötigten theoretischen Konzepte zu entwerfen und zu implementieren, und das so entstandene Dialogsystem zu evaluieren. Dazu wurde zunächst ein Katalog von Interaktionsmustern entworfen, der das Verhalten von den Dialogsystemen beschreiben soll. Desweiteren wurden Anwendungskriterien für jedes der beschriebenen Interaktionsmuster gefunden.

Das implementierte System wurde dann anhand des Kriteriums evaluiert, wie einfach neue Systeme unter Benutzung der entwickelten Architektur in verschiedenen Domänen und in verschiedenen Zielsprachen entworfen werden können.

Die Architektur ist hierarchisch in drei Schichten unterteilt. Die erste Schicht, genannt *Basis-Dialogdienste* bietet eine Reihe von zielsprach- und domänenunabhängigen Dialogdiensten an, wie zum Beispiel Satzanalyse oder Fragengenerierung. In der darüberliegenden Schicht, genannt *Benutzerschnittstellenschicht*, sind die generischen Interaktionsmuster realisiert. Die Komponenten der ersten Schicht sind parametrisiert durch zielsprach- und domänenabhängige Wissensquellen, wie zum Beispiel Ontologien, Grammatiken, und Aufgabenmodelle. Die Parametrisierung der den Interaktionsmustern zugrundeliegenden Dienste bewirkt die gewünschte Domänen- und Zielsprachunabhängigkeit. Vereinfachend gesagt, implementiert die zweite Schicht Äquivalente des in Abbildung 1.1 gezeigten Algorithmus, während die erste Schicht die von den Interaktionsmustern angeforderten Dienste implementiert. Die dritte Schicht, genannt *Dialog-Steuerungs-Schicht*, wählt die Instantiierungen der von der zweiten Schicht auszuführenden Algorithmen aus und übernimmt die Gesamtsystemsteuerung.

1.2.3 Überprüfung der These

Unter Benutzung dieses dreischichtigen Systems wurde dann ein Dialogsystem für ein interaktives Autonavigationssystem in Englisch und Deutsch entwickelt und evaluiert. Desweiteren wurden in einem Versuch von anderen Personen Dialogsysteme unter Benutzung der entwickelten Architektur entworfen und implementiert. Hierzu wurden die Wissensquellen, die von den Diensten in der unteren Schicht benutzt werden, an die Domäne und Zielsprache angepaßt.

1.2.4 Beiträge

Die Beträge dieser Arbeit sind in verschiedenen Kontexten zu sehen. Zum einen wurde ein existierender Formalismus zur Wissensrepräsentation, nämlich typisierte Merkmalsstrukturen, erweitert und an die Erfordernisse von generischen Dialogsystemen angepaßt. Die Erweiterungen umfassen objektorientierte Versionen und eine Variante dessen, was allgemeinhin als disjunktive Merkmalsstrukturen bekannt ist. Die Beiträge im Gebiet der typisierten Merkmalsstrukturen sind eher theoretischer Art, haben aber – wie auch deren Anwendung in der vorliegenden Systemimplementierung zeigt – praktische Bedeutung.

Auf dem Gebiet der Grammatikentwicklung wurde ein Verfahren zur Kombination von verschiedenen Komponenten der Grammatiken vorgeschlagen. Der gewählte Ansatz ermöglicht eine rasche prototypische Entwicklung von Grammatiken, die den Ansprüchen von Dialogsystemen entsprechen. Die gewählten Verfahren sind ähnlich zu aus der Literatur bekannten Prinzipien für komplexere Grammatiken wie zum Beispiel Hpsg oder Tag. Der Beitrag hier ist jedoch in dem Kontext zu sehen, daß zum einen einfachere Grammatikformalismen, teilweise sogar nur reguläre Grammatiken, im Bereich der Dialogverarbeitung dominieren. Zum anderen garantiert der vorgeschlagene Ansatz Typsicherheit der generierten Repräsentation, was ein wichtiges Merkmal für Dialogsysteme ist (für eine ausführliche Diskussion, siehe Kapitel 4).

Auf der Ebene der Dialogverarbeitung wird ein Katalog von Interaktionsmustern vorgeschlagen. Gegeben aufgabenorientierte Dialogsysteme, isoliert ein Interaktionsmuster ähnliche Verhaltensweisen der verschiedenen Dialogsysteme. Damit ist es möglich, eine generische Dialogstrategie getrennt von domänen- und sprachabhängigen Komponenten zu entwickeln. Anders herum ist es ebenfalls möglich, ein existierendes System schnell an eine andere Domäne oder Zielsprache lediglich durch die Anpassung der domänen- oder zielsprachspezifischen Komponenten vorzunehmen.

Desweiteren verdeckt ein Interaktionsmuster die Anbindung an die Anwendung. Korrekturen beispielsweise erfordern die Anpassung des Zustandes der Anwendung durch Invertierung von bereits ausgeführten Diensten. Interaktionsmuster bestimmen, in Abhängigkeit der im Diskurs vorhandenen Information, welche Dienste von der Anwendung angefordert und welche Dienste invertiert werden müssen.

Die auf den Interaktionsmustern aufbauende Architektur des Systems ermöglicht, daß Prototypen für neue Anwendungen schnell entwickelt werden können. Eine wesentliche Eigenschaft des vorgeschlagenen Ansatzes ist, daß

die so produzierten Prototypen zu voll funktionsfähigen Systemen ausgebaut werden können. Damit ist die Zeit, die in Entwicklung des Prototyps investiert worden ist, nicht verloren. Im Gegenteil wird im hier verfolgten Ansatz mit dem Prototyp der Grundstein zu einem vollständigen System gelegt. Ein wesentlicher Vorteil dieses Verfahren ist, daß somit Daten früh im Entwicklungsprozeß gesammelt werden können.

1.3 Andere Ansätze

Wie oben beschrieben, verfolgt der hier vorgestellte Arbeit einen objektorientierten, symbolischen Ansatz. Damit ist es möglich, die Komponenten des Dialogsystems einfach und effizient zu beschreiben. In der Literatur wurden eine Reihe von anderen Ansätzen zur Dialogverarbeitung beschrieben. Verschiedene symbolische Ansätze, deren Wissensquellen anders repräsentiert sind als in der vorliegenden Arbeit, zielen ebenfalls auf allgemeines Dialogmanagement. Statistische Ansätze versuchen durch das Erlernen von Dialogstrategien den Aufwand der Entwicklung zu reduzieren. Schließlich gibt es verschiedene Ansätze zum *Rapid Prototyping* von Dialogsystemen.

1.3.1 Symbolische Ansätze

Gegenwärtig verwenden die meisten natürlichsprachlichen Dialogsysteme die eine oder andere Form von rahmenbasierten Repräsentationen [Ward, 1994,Papineni *et al.*, 1999]. Dies ist teilweise begründet in der Tatsache, daß semantische Grammatiken [Ward, 1994,Gavalda, 2000] rahmenbasierte semantische Repräsentationen direkt aus den Parsebäumen erzeugen können. Darüberhinaus sind semantische Grammatiken robust gegenüber ungrammatikalischer Eingabe und einfach zu erstellen. Zusammen mit der Tatsache, daß rahmenbasierte Repräsentationen einfach zu implementieren sind, hat dies zu einer weiten Verbreitung dieses Ansatzes geführt.

Um die hohe Robustheit der semantischen Satzanalyse zu garantieren, weisen semantische Parser den Eingaben nur begrenzt syntaktische Struktur zu. Als Konsequenz sind die resultierenden semantischen Repräsentationen nicht invariant gegenüber Reformulierungen. Mit anderen Worten, semantische Grammatiken sind häufig so geschrieben, daß zwei Sätzen mit derselben Bedeutung, aber unterschiedlicher syntaktischer Struktur, unterschiedliche Repräsentationen zugewiesen werden.[1] Dies ist in Dialogsystemen unerwünscht, da der interne Zustand von der im Diskurs etablierten Information, aber nicht dessen Oberflächenform abhängen soll. Dieses Problem kann durch das Nachschalten eines Nachverarbeitungsschrittes umgangen werden, wie es beispielsweise in der Maschinellen Übersetzung [Levin *et al.*, 1998,Woszczyna *et al.*, 2000] getan wird. Vom Standpunkt des *Rapid Prototyping* sind diese Nachverarbeitungsschritte

[1] Zwar ist die Zuweisung von Repräsentationen nur von der Grammatik abhängig. Dies bedeutet, daß semantische Grammatiken so geschrieben werden können, daß Sätze mit derselben Bedeuting dieselbe Repräsentation zugewiesen werden. Dies stellt jedoch zusätzliche Anforderungen an die Grammatikentwicklung, die in der Praxis häufig nicht erfüllt werden können.

jedoch unerwünscht, da sie die Komplexität der Spezifikation erhöhen (zusätzlich zu den syntaktisch-semantischen Grammatikregeln müssen Konvertierungsregeln angegeben werden).

Nachdem die semantischen Repräsentationen der Äußerungen erzeugt worden sind, können die Prinzipien von Abella und Gorin [1999] für das Dialogmanagement angewandt werden. Der dort vorgeschlagene Ansatz ist, ähnlich wie die vorliegende Arbeit, *informationsbasiert* in dem Sinne, daß Dialogzustände durch die Spezifizität der vorhanden Information beschrieben werden. Ein Unterschied zu der vorliegenden Arbeit ist, daß in [Abella and Gorin, 1999] keine generische Möglichkeit angegeben ist, den Inhalt von Klärungsfragen zu bestimmen. Damit ist es schwierig, auf den vorgestellten Konzepten eine generische Dialogstrategie zu entwickeln. Darüber hinaus ist nicht angegeben, wie Dienstanforderungen an die Anwendung auszuführen und zu invertieren sind. Interaktionen mit der Anwendung sind in Rudnicky und andere [1999] beschrieben, wo sogenannte *handler* die Kontrolle an die Anwendung abgeben, wenn bestimmte Information im Diskurs etabliert ist. Die im Zuge dieser Arbeit entwickelten objektorientierten typisierten Merkmalsstrukturen sind eine andere Lösung desselben Problems. Objektorientierte typisierte Merkmalsstrukturen implementieren Dienstanforderungen an die Anwendung durch eine Verallgemeinerung einer Typinferenzprozedur und garantieren damit darüber hinaus Typsicherheit der Repräsentationen, was bei Rudnicky und andere [1999] nicht gegeben ist.

Viele Dialogsysteme verwenden die eine oder andere Form von Skriptsprachen, um die Interaktion der beteiligten Komponenten zu steuern [Levin and Pieraccini, 1999,Papineni *et al.*, 1999]. Alternativ dazu können deklarative Programmiersprachen wie ProLog für denselben Zweck verwendet werden. Die Arbeit von Smith [Smith, 1992] zeigt, wie der Backtracking von ProLog ausgenutzt werden kann, um fehlende Information zu bestimmen. Fehlende Information ist durch das Fehlschlagen einer ProLog-Anfrage gekennzeichnet. Dadurch wird der semantische Inhalt von Klärungsfragen bestimmt. Je nach Entwurf der Steuerungskomponente können die Skripten domänenabhängig oder -unabhängig sein.

France Télécom [Bretier and Sadek, 1996,Sadek *et al.*, 1997] entwickelte ein natürlichsprachliches Dialogsystem namens Artimis dessen Dialogmanager auf den Prinzipien des *Rationalen Verhaltens* (engl. *Rational Agency*) beruht. Das Verhalten des Dialogmanagers ist durch in der Modallogik S45 formulierte Regeln ausgedrückt. Dies ist ein sehr mächtiger Ansatz, der es erlaubt, über verschiedene Domänen zu generalisieren.

1.3.2 Statistische Ansätze

Die Forschungsgruppen bei At & t und Lucent haben in den letzten Jahren verschiedene Verfahren zur statistischen Verarbeitung von aufgabenorientierten Dialogen vorgeschlagen.

Gorin et al [1997] stellen ein System zur Anrufweiterleitung vor. Die Aufgabe des Dialogsystems ist es, Anrufer an eine von 15 Abteilung weiterzuleiten, wo dann das Anliegen des Anrufers bearbeitet werden kann (beispielsweise das

Erfragen von Gebührentarifen, Probleme mit Rechnungen und so weiter). Diese Arbeit zeichnet sich dadurch aus, daß die Anrufer auf den offenen Systemprompt *"How may I help you?"* antworten können. Das Problem der Dialogverarbeitung ist somit in erster Iteration als Klassifikation des Anrufes in eine von 15 Klassen zu sehen. Weitere Arbeiten [Wright *et al.*, 1998] jedoch ergänzen diesen Ansatz um aufgabenorientierte Unterdialoge, die fehlende Information vom Anrufer erfragen können. Die Offenheit des Systemprompts impliziert natürlich Probleme für die Spracherkennung und Verarbeitung. Die vorgeschlagene Lösung sieht vor, bedeutungstragende Satzfragmente (engl. *salient expressions*) zur Bestimmung der Intention des Benutzers heranzuziehen. Bedeutungstragende Satzfragmente werden durch informationstheoretische Maße bestimmt [Gorin, 1996], wobei das Maß die Information des Satzfragmentes relativ zur Klassifikation angibt. Das Sprachmodell des verwendeten Spracherkenners verfügt ebenfalls über Kenntnis der bedeutungstragenden Satzfragmente, um ausreichend gute Erkennung der bedeutungstragenden Elemente sicherzustellen.

In den Arbeiten von Chu-Carroll und Carpenter [1998, 1999] wird eine alternative Lösung für dasselbe Problem vorgeschlagen. Hier werden anstelle informationstheoretischen Maßen Uni-, Bi- und Trigramme verwendet, um bedeutungstragende Einheiten für die Klassifikation des Anrufes zu beschreiben. Prototypen von Anrufen werden dann durch vektorielle Repräsentationen ihrer bedeutungstragenden Einheiten repräsentiert. Die Klassifikation von Anrufen erfolgt dann durch die Umwandlung des Anrufes in dieselbe vektorielle Repräsentation und anschließendes Anwenden von aus dem *Information Retrieval* bekannten Bewertungstechniken.

Singh et al [2002] wenden Techniken des Verstärkungslernen (engl. *reinforcement learning*) an, um Dialogstrategien zu lernen. In ihrem System NJFun können sich Anwender Auskünfte zu Vergnügungen in New Jersey geben lassen. Der Dialogzustand ist dabei repräsentiert durch einen aufgabenabhängigen Zustandsvektor. Aufgabe des Lernverfahrens ist es, den Folgezustand für einen gegebenen Dialogzustand zu bestimmen. Dazu wird am Ende jeden Dialoges eine Rückmeldung über die Qualität des Dialoges an das Lernsystem gegeben. Das Verstärkungslernen "verteilt" die Rückmeldung dann auf einzelne Zustandsübergänge, die während des Dialoges getätigt worden sind.

Während Singh et al [2002] die Dialogstrategie durch einen Markov-Entscheidungsprozeß modellieren, und somit verdeckte Zuständen oder die teilweise Beobachtbarkeit des tatsächlichen Prozesses nicht in Betracht ziehen, schlagen Roy et al [2000] einen auf partiell beobachtbaren Markov-Entscheidungsprozessen (POMDP) basierenden Dialogmanager vor. Das Problem bei partiell beobachtbaren Entscheidungsproblemen ist jedoch der hohe Rechenaufwand.

Zusätzlich zur robusteren Verarbeitung von Daten ist der Vorteil von statistischen Ansätzen zur Dialogverarbeitung, daß domänspezifisches oder auch linguistisches Wissen gelernt werden kann. Damit ist weniger Expertenwissen zur Erstellung eines Dialogsystems erforderlich. Alle beschriebenen Ansätze benötigen jedoch teilweise erhebliche Mengen an Daten, um erfolgreich zu sein. Im Falle großer Telephongesellschaften stellt dies nicht notwendigerweise ein Problem dar, da Anrufe an existierende *Call Center* aufgenommen und transkri-

biert werden können. Die vorliegende Arbeit versucht das zur Erstellung eines Systems benötige Expertenwissen durch Separierung von Entwurfsentscheidungen zu reduzieren. Damit können mehrere Teile existierender Dialogsysteme wiederverwendet werden. Eine Bereitstellung eines Systems in einer neuen Anwendungsdomäne erfordert dann lediglich das Anpassen der Komponenten, die sich von einer existierenden Implementierung unterscheiden.

1.3.3 Ansätze zum Rapid Prototyping

Der schnelle Entwurf von interaktiven Dialogsystemen ist seit kurzem Gegenstand der Forschung geworden. Zum Teil begründet durch den zunehmenden Bedarf von sprachverstehenden Anwendungen und Telephonie-basierten Diensten, ist die Anzahl eingesetzter Systeme gestiegen. Zu diesem Zeitpunkt können zum Entwurf dieser Systeme drei Typen von Ansätzen unterschieden werden. Ein erster Ansatz zum schnellen Entwurf für Dialogsysteme besteht aus graphischen Editoren, mit deren Hilfe man den Dialogfluß durch endliche Automaten spezifizieren kann [Sutton *et al.*, 1996,Cole, 1999]. Diese Systeme setzen einen Dialog mit einem Pfad vom Anfangszustand zu einem der Endzustände des Automaten gleich. Mögliche Aktionen des Systems sind durch Annotationen der Zustandsübergänge spezifiziert. Problematisch ist dieser Ansatz dadurch, daß endliche Automaten nur schwer verschiedene Aspekte der Dialogspezifikation separieren können (siehe auch dazu in Kapitel 6 und 7 für eine ausführlichere Diskussion). Zum Beispiel müssen Bestätigungsfragen, zu stellen bei geringen Konfidenzmaßen der Spracheingabe, jedem Zustand, in dem sie gestellt werden sollen, zugeordnet werden. Es ist nicht möglich, das Verhalten des Systems durch eine allgemeingültige Regel der Form *"Wenn immer das Konfidenzmaß der Erkenners unter 50% liegt, soll der Benutzer zur Wiederholung der Eingabe aufgefordert werden"* zu beschreiben.

Ein anderer Ansatz zur Wiederverwendung von Dialogkomponenten, genannt *Speech Objects* wurde von der kalifornischen Firma *Nuance* entwickelt [Andersson *et al.*, 2001]. Hier wird die Eingabe von domänenspezifischen Daten mittels VoiceXML über Java-Objekte vorgenommen. So gibt es beispielsweise ein *Speech Object* für die Eingabe eines Datums, ein anderes für die Eingabe von Telephonnummern und so weiter. Gemeinsame Funktionalität der *Speech Objects*, wie zum Beispiel Bestätigungsfragen, kann durch die Erweiterung der Basisklasse implementiert werden. Auf der anderen Seite kann die Dialogstrategie immer nur lokal agieren: das System befindet sich solange in einem Zustand, wie das *Speech Object* aktiv ist. Flexiblerer Dialog mit Gegenfragen zum Beispiel durch *Speech Objects* zu beschreiben ist schwierig.

Ein dritter Ansatz besteht im Entwurf von Bibliotheken von wiederverwendbaren Dialogkomponenten. Auf der Beobachtung basierend, daß das Verhalten des Dialogsystems in sich ähnelnden Situationen vorhersagbar sein soll, entwickeln Araki et al [1999] eine Bibliothek von wiederverwendbaren Dialogstrategien. [Kölzer, 1999] schlägt eine Dialogsystemarchitektur vor, wobei die Systemkomponenten durch einen graphischen Editor spezifiziert werden können.

1.4 Übersicht der Ausarbeitung

Diese Ausarbeitung ist in drei Teile unterteilt. Im ersten Teil, in den Kapiteln 2 bis 5, wird die Funktionalität der *Basis-Dialogdienste* beschrieben. Kapitel 2 beschreibt die verwendeten typisierten Merkmalsstrukturen und zwei wichtige Generalisierungen. Kapitel 3 beschäftigt sich damit, wie mehrere typisierte Merkmalsstrukturen kompakt abgespeichert, und, als Konsequenz daraus, ihre informationellen Unterschiede einfach ersehen und quantifiziert werden können. Die Existenz oder Nichtexistenz der informationellen Unterschiede steuern die generischen Interaktionsmuster. In Kapitel 4 werden Grammatiken, Dienste zur Satzanalyse und zur schablonenbasierten Generierung beschrieben. Kapitel 5 schließlich beschreibt Dienste, die Datenbankzugriff, Diskursanpassung und Kontrolle über das Aufgabenmodell des Dialogsystems ermöglichen.

Der zweite Teil beschäftigt sich mit dem Konzept von generischen Interaktionsmustern. In Kapitel 6 werden abstrakte Dialogzustände definiert, die den Fortschritt des Dialogs zielsprachen- und domänenunabhängig beschreiben. Kapitel 7 führt dann den Begriff des generischen Interaktionsmusters ein. Ein generisches Interaktionsmuster kann betrachtet werden als das funktionale Äquivalent eines Zustandsübergangs zwischen zwei abstrakten Dialogzuständen. Die folgenden Kapitel 8, 9, 10 und 11 beschäftigen sich im generischen Interaktionsmustern zum Erfragen, Zurücknahmen und Navigieren von Information. Kapitel 12 gibt eine Übersicht der vorgestellten Interaktionsmuster.

Der dritte Teil beschreibt die Auswertung der Experimente. Kapitel 13 beschreibt die Auswertung des *Rapid Protoyping* Experiments. Kapitel 14 schließlich faßt die erzielten Ergebnisse zusammen und diskutiert zukünftige Arbeit.

Teil I

Basis-Dialog-Dienste

Kapitel 2

Typisierte Merkmalsstrukturen und Erweiterungen

Interaktionen von Benutzern mit natürlichsprachlichen Dialogsystemen erstrecken sich typischerweise über mehrere Äußerungen oder *Turns*. Der Informationsgehalt der während des Dialogs konstruierten Repräsentationen wird durch die Ergebnisse von Datenbankanfragen oder durch die Integration von Antworten auf Klärungsfragen weiter verändert. Das inkrementelle Konstruieren der Repräsentationen erfordert folglich, daß die folgenden Operationen auf Repräsentationen ausgeführt werden können:

(i) **Vergleich bezüglich des Informationsgehaltes**
zum Beispiel um zu bestimmen, ob die im Diskurs vorhandene Information ausreichend ist, die vom Benutzer gewünschte Aktion auszuführen,

(ii) **Unifikation von Repräsentationen**
zum Beispiel um eine Antwort auf eine Klärungsfrage in den Diskurs zu integrieren,

(iii) **Konsistenzüberprüfung**
zum Beispiel um zu bestimmen, ob die vom Benutzer übermittelte Information widersprüchlich ist oder nicht,

(iv) **Zurücknahme von Repräsentationen**
zum Beispiel um falsch verstandene Information wieder aus dem Diskurs zu entfernen

Es ist damit keine Überraschung, daß natürlichsprachlich Dialogsysteme in der Lage sein müssen, *partielle* Information verarbeiten zu können. Dieser Anforderung ist meist durch den Einsatz rahmenbasierter Repräsentationen Rechnung getragen [Bilange *et al.*, 1990,Bilange, 1991,Bobrow and Group, 1977]. Des weiteren werden rahmenbasierte Repräsentationen von verschiedenen robusten Parsern erzeugt, so zum Beispiel vom Parser PHOENIX [Ward, 1994] und dem in dieser Arbeit verwendeten Parser SOUP [Gavalda, 2000].

Im Bereich der Künstlichen Intelligenz wurden Zusammenhänge zwischen rahmenbasierten Repräsentationen und Beschreibungslogiken festgestellt. Be-

schreibungslogiken können aufgefaßt werden als „Designer-Logiken“, deren Ausdrucksstärke auf die zu zu modellierenden Sachverhalte angepaßt werden kann. Hierbei muß ein Kompromiß zwischen Ausdrucksstärke der Logik und der Effizienz der zugrundeliegenden Inferenzalgorithmen geschlossen werden. Beschreibungslogiken habe in letzter Zeit vielfältige Anwendung gefunden, unter anderem auch in natürlichsprachlichen Dialogsystemen [Ludwig *et al.*, 1998].

Im Bereich der Computerlinguistik haben Varianten von Beschreibungslogiken, die *Merkmalslogiken*, besondere Beachtung gefunden. Die zu modellierenden Sachverhalte sind hier im wesentlichen Bereiche aus der Syntax [Pollard and Sag, 1994]. Arbeiten von Nebel und Smolka [1991] haben das Verhältnis zwischen Merkmalslogiken und Beschreibungslogiken herausgearbeitet, das vereinfachend paraphrasiert werden kann wie folgt: Merkmalslogiken sind Beschreibungslogiken, wobei die Rollen der Merkmalslogiken (Merkmale) maximal einen Füller aufnehmen können.

Eine besondere Variante der Merkmalslogiken ist die typisierte Merkmalslogik entwickelt von Carpenter [1992]. Die Besonderheit hier ist, daß die Typen in einer Vererbungshierarchie, ähnlich wie in objekt-orientierten Programmiersprachen angeordnet werden können. Damit lassen sich Sachverhalte einfach und objekt-orientiert modellieren. Aus diesem Grunde wurde diese Logik als Basis für das im Rahmen dieser Arbeit entwickelte Dialogsystem gewählt.

2.1 Einleitung

Die von Carpenter entwickelte typisierte Merkmalslogik (und deren Terme, graphisch dargestellt als typisierte Merkmalsstrukturen) zeichnen sich durch die Typeinschränkungen der Terme aus. In der Computerlinguistik wird dies in erster Linie verwendet, um syntaktische Sachverhalte zu beschreiben. Zum Beispiel kann im Deutschen Subjekt-Verb Kongruenz durch ein Merkmalsbündel bestehend aus den Merkmalen PERSON und NUMBER beschrieben werden. Das Merkmalsbündel hat den Typ *agr* (Agreement), und man sagt, daß die Merkmale PERSON und NUMBER für den Typ *agr* erlaubt sind.

Im Rahmen von natürlichsprachen Dialogsystemen ist die syntaktische Modellierung weniger relevant, allerdings können die semantischen Sachverhalte der zugrundeliegenden Anwendung sehr einfach durch die Vererbungshierarchie der Typen beschrieben werden. Die Idee ist hier, die semantische Repräsentationen der natürlichsprachlichen Anfragen in partiellen Merkmalsstrukturen zu repräsentieren. Die so erstellten Beschreibungen der erwähnten Objekte, deren Eigenschaften, Aktionen und Zustände können dann im Laufe des Dialogs mit weiterer Information (zum Beispiel aus Datenbanken) ergänzt werden.

Unter anderem sind auf typisierten Merkmalsstrukturen Subsumptions- und Unifikationsoperationen definiert. Diese Operationen erfüllen somit die oben gestellten Bedingungen (i) und (ii). Zusätzlich ist es möglich, auf typisierten Merkmalsstrukturen verschiedene Arten von Typinferenzen auszuführen. In diesem Zusammenhang kann eine Typinferenz vereinfachend als eine Operation beschrieben werden, die die Merkmalsstruktur um statische Information aus der Vererbungshierarchie anreichert. Sollte dabei eine inkonsistente Merkmalsstruktur entstehen, ist die Information der Merkmalsstruktur nicht konsistent mit der

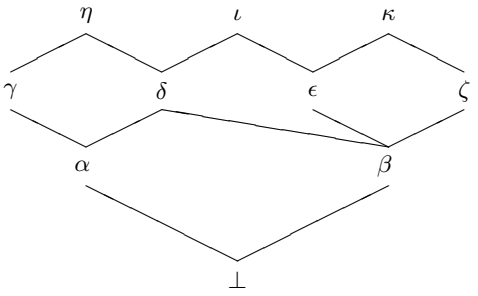


Abbildung 2.1. Eine Typenhierarchie

Domänenmodellierung. Diese Operation erfüllt so die Bedingung (iii). Auf der anderen Seite gibt es keine Operation die Bedingung (iv) erfüllt. Im folgenden Kapitel wird deswegen eine solche Operation eingeführt.

In diesem Kapitel werden die Merkmalsstrukturen in zwei Richtungen verallgemeinert. Die erste Erweiterung erlaubt die Spezifikation von *Methoden* in der Vererbungshierarchie und ermöglicht es so dem Dialogsystem, Dienste in der Anwendung abhängig vom Informationsgehalt der Repräsentationen aufzurufen. Die zweite Erweiterung ermöglicht es, neben der (semantischen) Typinformation auch weitere Informationen, die relevant für die Dialogverarbeitung sind, abzulegen. Dazu zählen unter anderem Konfidenzinformationen des Spracherkenners.

2.2 Die Typenhierarchie

Die in diesen Abschnitten angegebenen Definitionen und Theoreme sind aus [Carpenter, 1992] entnommen.

Typisierte Merkmalsstrukturen sind Graphstrukturen, deren Knoten ein Typsymbol aufweisen, und deren Kanten mit einem Merkmalssymbol gekennzeichnet sind. Die Typen partiell geordnet; die partielle Ordnung ist im besonderen eine untere Halbordnung.

Definition 1 (Typenhierarchie [Carpenter, 1992]). *Eine* Typenhierarchie *ist eine endliche beschränkte vollständige partielle Halbordnung.* $\langle \mathsf{Type}, \sqsubseteq \rangle$.

Die hier verwendete Notation folgt der Carpenters in dem der Informationsgehalt in den Typenhierarchien von unten nach oben wächst. Es folgt aus der Definition, daß mit der Existenz einer oberen Schranke für eine zwei Typen τ, ρ auch eine eindeutige kleinste obere Schranke für τ, ρ existiert. Damit kann man sagen, daß die Unifikation der Typen τ und ρ die kleine obere Schranke von τ und ρ ist, falls sie existiert. Anderenfalls ist die Unifikation undefiniert. Parallel zu der Unifikation ist die Generalisierung von zwei Typen τ und ρ definiert als die größte untere Schranke von τ und ρ. Abbildung 2.1 zeigt eine Typenhierarchie, die auch in den folgenden Beispielen weiter verwendet wird.

Die Relation $\sqsubseteq$ bezeichnet die Subsumptionsrelation. In Analogie führen wir die folgenden Symbole ein: die Relation $\sqsubset$ hält zwischen zwei Typen τ und ρ wenn $\tau \sqsubseteq \rho$ und $\tau \neq \rho$. Desweiteren bedeutet $\tau \dot{\sqsubset} \rho$ die Relation zwischen zwei Typen τ und ρ wo es keinen weiteren Typen $\omega \neq \tau, \rho$ gibt so daß $\tau \sqsubset \omega \sqsubset \rho$ gilt.

2.3 Typisierte Merkmalsstrukturen

In diesem Abschnitt werden weitere relevante Begriffe der Merkmalsstrukturen aus [Carpenter, 1992] eingeführt. Eine typisierte Merkmalsstruktur kann betrachtet werden als ein gewurzelter Graph, dessen Knoten mit den Typen aus einer gegebenen Typhierarchie $\langle \mathsf{Type}, \sqsubseteq \rangle$ und dessen Kanten mit Merkmalssymbolen aus einer Menge von Merkmalen Feat annotiert sind.

Definition 2 (Typisierte Merkmalsstrukturen[Carpenter, 1992]). *Eine* typisierte Merkmalsstruktur *über* Feat *und* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *ist ein Tupel* $F = \langle Q, \bar{q}, \theta, \delta \rangle$*, wobei:*

(i) Q *ist eine endliche Menge Knoten,*
(ii) $\bar{q}$ *ist die Wurzel,*
(iii) $\theta : Q \rightarrow \mathsf{Type}$ *ist eine totale Funktion, die Knoten auf deren Typen abbildet,*
(iv) $\delta : \mathsf{Feat} \times Q \rightarrow Q$ *ist eine partielle Merkmalsfunktion, die einen Knoten und ein Merkmal auf einen eindeutig bestimmten Folgeknoten abbildet.*

Eine Merkmalsstruktur heißt atomar*, wenn Q nur einen Knoten enthält. Die Menge $\mathcal{F}$ bezeichnet die Menge aller Merkmalsstrukturen.*

Im folgenden wird angenommen, daß der der Merkmalsstruktur zugrundeliegende Graph azyklisch ist. In praktischen Anwendungen der Merkmalsstrukturen in Dialogsystem ist es sogar ausreichend, die zugrundeliegenden Graphen auf Bäume zu beschränken. Abbildung 2.2 zeigt drei Merkmalsstrukturen konstruiert über der Typenhierarchie in Abbildung 2.1 und den Merkmalen $\mathsf{Feat} = \{f, g, h, i\}$.

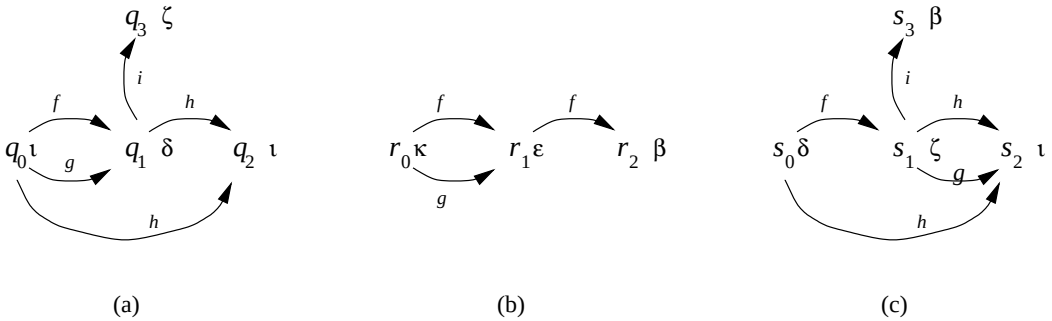


Abbildung 2.2. Drei typisierte Merkmalsstrukturen

Subsumption von typisierten Merkmalsstrukturen ist die Verallgemeinerung der partiellen Ordnung über Typen zu Merkmalsstrukturen. Die Aussage, daß

Struktur F Struktur F' subsumiert, bedeutet anschaulich, daß Struktur F' alle Information enthält, die auch F enthält (und eventuell mehr).

Definition 3 (Subsumption [Carpenter, 1992]). *Eine Merkmalsstruktur $F = \langle Q, \bar{q}, \theta, \delta \rangle$ subsumiert eine Merkmalsstruktur $F' = \langle Q', \bar{q}', \theta', \delta' \rangle$, $F \sqsubseteq F'$ genau dann, wenn es eine totale Funktion $h : Q \to Q'$ gibt, sodaß folgende Bedingungen gelten:*

1. $h(\bar{q}) = \bar{q}'$,
2. $\theta(q) \sqsubseteq \theta'(h(q))$ *für alle* $q \in Q$,
3. $h(\delta(f,q)) = \delta'(f, h(q))$ *für alle* $q \in Q$ *und Merkmale* f, *für die* $\delta(f,q)$ *definiert ist.*

Die erste Bedingung in obiger Definition stellt sicher, daß die Wurzelknoten der den Strukturen zugrundeliegenden Graphen aufeinander abgebildet werden. Die zweite Bedingung fordert, daß, wenn ein Knoten q auf einen Knoten q' abgebildet wird, der Informationsgehalt von q' wenigstens dem Informationsgehalt von q entspricht. Die dritte Bedingung stellt sicher, daß eine Merkmalsdefinition in F auch in F' definiert ist.

Unifikation von Merkmalsstrukturen ist der Prozeß der Informationskonjunktion. Um zwei Strukturen zu unifizieren, wird eine Äquivalenzrelation über die Menge der Knoten beider Strukturen konstruiert. Die Wurzelknoten sind äquivalent; desweiteren sind alle Knoten miteinander äquivalent, die von äquivalenten Knoten über ein Merkmal f erreicht werden können. Jeder Äquivalenzklasse wird die kleinste obere Schranke der Typen der in der Äquivalenzklasse enthaltenen Knoten zugewiesen (Typunifikation). Sollte eine solche Schranke nicht existieren, ist die Unifikation der Merkmalsstrukturen nicht definiert. Eine formale Definition der Unifikation folgt.

Definition 4 (Unifikation [Carpenter, 1992]). *Seien F, F' Merkmalsstrukturen $F \sim \langle Q, \bar{q}, \theta, \delta \rangle$ und $F' \sim \langle Q', \bar{q}', \theta', \delta' \rangle$ so daß $Q \cap Q' = \emptyset$. Die Äquivalenzrelation $\bowtie$ über $Q \cup Q'$ ist definiert als die kleinste Äquivalenzrelation so daß*

1. $\bar{q} \bowtie \bar{q}'$
2. $\delta(f,q) \bowtie \delta(f,q')$ *wenn beide definiert sind und* $q \bowtie q'$

Die Unifikation *von F und F' ist dann definiert als*

$$F \sqcup F' = \langle (Q \cup Q')/_{\bowtie}, [\bar{q}]_{\bowtie}, \theta^{\bowtie}, \delta^{\bowtie} \rangle$$

wobei

$$\theta^{\bowtie}([q]_{\bowtie}) = \bigsqcup \{(\theta \cup \theta')(q') \mid q' \bowtie q\}$$

und

$$\delta^{\bowtie}(f, [q]_{\bowtie}) = \begin{cases} [(\delta \cup \delta')(f,q)]_{\bowtie} & \textit{falls } (\delta \cup \delta')(f,q) \textit{ wohldefiniert ist} \\ undefiniert & anderenfalls. \end{cases}$$

falls alle oberen Schranken in der Konstruktion von $\theta^{\bowtie}$ existieren. Anderenfalls ist $F \sqcup F'$ undefiniert.

Carpenter zeigt, daß die Unifikation von zwei Merkmalsstrukturen, falls sie existiert, durch die Subsumptionsordnung definiert werden kann. $F \sqcup F'$ ist

nämlich die kleinste obere Schranke von F und F' bezüglich der Subsumptionsrelation.

2.4 Wohltypisierte Merkmalsstrukturen und Typinferenz

Die Definition 2 für Merkmalsstrukturen schränkt in keiner Weise die Kombination von Typen und Merkmalen ein. Ein Merkmal kann für einen Knoten beliebigen Typs definiert sein. Als Beispiel sei eine Typenhierarchie mit den Typen *obj_restaurant* und *obj_hotel* und die Merkmale TYPEOFFOOD und PRICEPERNIGHT gegeben. Die Merkmalsstrukturen in Abbildung 2.3 (a) stellen zwei gültige Strukturen dar, obwohl die repräsentierte Information, von einem semantischen Standpunkt, keinen Sinn macht.

$$\begin{bmatrix} \textit{obj\_restaurant} \\ \text{PRICEPERNIGHT } \textit{\$70} \end{bmatrix} \begin{bmatrix} \textit{obj\_hotel} \\ \text{TYPEOFFOOD } \textit{italian} \end{bmatrix}$$

(a)

$$\begin{bmatrix} \textit{obj\_restaurant} \\ \text{TYPEOFFOOD } \textit{italian} \end{bmatrix} \begin{bmatrix} \textit{obj\_hotel} \\ \text{PRICEPERNIGHT } \textit{\$70} \end{bmatrix}$$

(b)

Abbildung 2.3. Semantisch nicht plausible (a) und semantisch plausible (b) partielle Repräsentationen von zwei Objekten.

Um Strukturen bezüglich ihrer Konsistenz mit Weltwissen zu testen, gibt es *Typinferenzprozeduren*. Eine Typinferenzprozedur kombiniert Information von sogenannten *Wohlgeformtheitsspezifikationen* in die Merkmalsstrukturen.

Definition 5 (Wohlgeformtheitsspezifikationen[Carpenter, 1992]). *Eine* Wohlgeformtheitsspezifikation *über der Merkmalshierarchie* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *und einer Menge von Merkmalen* Feat *ist gegeben durch eine partielle Funktion* $Approp : \mathsf{Type} \times \mathsf{Feat} \rightarrow \mathsf{Type}$ *welche die folgenden zwei Bedingungen erfüllt:*

(i) **Merkmalseinführung**
für jedes Merkmal $f \in \mathsf{Feat}$ *gibt es einen allgemeinsten Typ* $Intro(f) \in \mathsf{Type}$ *so daß* $Approp(Intro(f), f)$ *wohldefiniert ist, und*

(ii) **Monotonie/Aufwärts-Abgeschlossenheit**
wenn $Approp(\sigma, f)$ *wohldefiniert ist und* $\sigma \sqsubseteq \tau$*, dann ist auch* $Approp(\tau, f)$ *wohldefiniert und erfüllt* $Approp(\sigma, f) \sqsubseteq Approp(\tau, f)$

Informell ausgedrückt bedeutet dies, daß eine Merkmalsstruktur die Wohlgeformtheitsspezifikationen erfüllt, wenn die in den Merkmalsstrukturen enthaltene Information wenigstens so spezifisch wie die in den Wohlgeformtheitsspezifikationen ist.

Die Funktion *Approp* kann wie folgt auf Merkmalspfade π der Länge ≥ 1 erweitert werden.

$$Approp(\sigma, f_1 \mid \pi) = \begin{cases} Approp(Approp(\sigma, f_1), \pi) & \text{falls } Approp(\sigma, f_1) \text{ und} \\ & Approp(Approp(\sigma, f_1), \pi) \\ & \text{definiert sind,} \\ \text{undefiniert} & \text{anderenfalls} \end{cases}$$

Definition 6 (Wohltypisierte Merkmalsstrukturen [Carpenter, 1992]). *Eine Merkmalsstruktur $F = \langle Q, \bar{q}, \theta, \delta \rangle$ wird* wohltypisiert *genannt, wenn für alle q und f, für die $\delta(q, f)$ definiert ist, $Approp(\theta(q), f)$ ebenfalls definiert ist, und desweiteren $Approp(\theta(q), f) \sqsubseteq \theta(\delta(q, f))$ gilt.*

Im obigen Beispiel werden beispielsweise die Merkmalsstrukturen in Abbildung 2.3 (a) durch die folgenden Wohlgeformtheitsspezifikationen herausgefiltert, während die Merkmalsstrukturen in 2.3 (b) den Spezifikation entsprechen.

$$\begin{aligned} Intro(\text{PRICEPERNIGHT}) &= obj\_hotel \\ Intro(\text{TYPEOFFOOD}) &= obj\_restaurant \\ Approp(obj\_hotel, \text{PRICEPERNIGHT}) &= currency \\ Approp(obj\_restaurant, \text{TYPEOFFOOD}) &= nationality \end{aligned}$$

Eine Prozedur, die diese Typüberprüfungen vornimmt, wird *Typinferenzprozedur* genannt und zeigt die folgenden Eigenschaften:

Definition 7 (Inferenzprozedur[Carpenter, 1992]). *Eine partielle Funktion $I : D \rightarrow D$ über einer partiellen Ordnung $\langle D, \leq \rangle$ wird* Inferenzprozedur *genannt wenn für alle $d, d' \in D$ die folgenden Eigenschaften gelten:*

(i) **Monotonie**
wenn $d \leq d'$ und $I(d')$ wohldefiniert ist, dann ist $I(d)$ ebenfalls wohldefiniert; außerdem gilt $I(d) \leq I(d')$,

(ii) **Steigend**
$d \leq I(d)$ wenn $I(d)$ wohldefiniert ist, und

(iii) **Idempotent**
$I(I(d)) = I(d)$ wenn $I(d)$ wohldefiniert ist.

Diese Definition beendet die Einführung in typisierte Merkmalsstrukturen. Alle bis hier angegebenen Definitionen sind aus Carpenter [1992] entnommen. Alle folgenden Definitionen sind eigenständige Generalisierungen der typisierten Merkmalsstrukturen.

2.5 Objekt-Orientierte Erweiterungen

Dialogsysteme müssen die Fähigkeit haben, zu jeder Zeit mit der Anwendung kommunizieren zu können. Zum Beispiel müssen Visualisierungen dem internen Zustand des Diskurses angepaßt werden. Wenn beispielsweise der Benutzer eines Stadtinformationskiosks sich nach Hotelreservierungen erkundigt, sollten

die den Anforderungen entsprechenden Hotels auf einer Straßenkarte angezeigt werden können. Als weiteres Beispiel sei ein sprachverstehender Videorekorder genannt: wenn sich der Benutzer nach der Länge eines Filmes erkundigt, und die Datenbank Start- und Endzeit des Films in den Diskurs einbringt, muß das Dialogsystem eine Funktion in der Anwendung aufrufen, die daraus die Länge des Filmes errechnet. Abbildung 2.4 stellt diesen Sachverhalt dar.

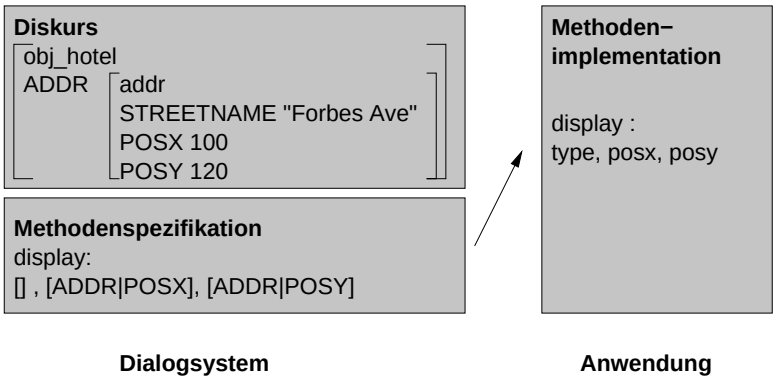


Abbildung 2.4. Aufruf von Funktionen in der Anwendung

Für eine portable Beschreibung des Verhaltens eines Dialogsystems ist es angebracht, den Zeitpunkt einer Dienstanforderung von der im Diskurs verfügbaren Information abhängig zu machen. Damit wird die Entscheidung, einen Dienst anzufordern, von der Dialogstrategie entkoppelt.[1] Im Falle des Videorekorders ist es damit unerheblich, ob die Information zur Anfangszeit des Filmes aus einer Datenbank stammt oder vom Anwender explizit erwähnt worden ist. Dies ist ein weiteres Beispiel für das Entkoppeln unabhängiger Entwurfsentscheidungen.

Um eine Dienstanforderung in Abhängigkeit der verfügbaren Information zu ermöglichen, wird eine Erweiterung der Wohlgeformtheitsspezifikation auf Methodenspezifikationen vorgeschlagen. Die Methodenspezifikationen können als Constraints über den Informationsgehalt der assoziierten Merkmalsstrukturen betrachtet werden. Ein Methodenaufruf beschreibt dabei eine Dienstanforderung des Dialogsystems an die Anwendung. Eine Methode wird aufgerufen, wenn der ihr entsprechende Constraint erfüllt ist. Im folgenden wird eine allgemeine Prozedur vorgestellt, die bestimmt, welche Methoden aufgerufen werden sollen. Weiterhin wird gezeigt, daß diese Prozedur eine Inferenzprozedur ist. Die Methodenspezifikationen sind den von Rudnicky und Wu [1999] vorgeschlagenen *Handler* ähnlich.

[1] In auf endlichen Automaten basierenden Dialogsystemem beispielsweise ist das nicht der Fall. Aktionen des Systems sind mit Zustandsübergängen, und damit mit der Dialogstrategie, verkoppelt.

2.5.1 Methodenspezifikationen

Die Typenhierarchie, die im Dialogsystem verwendet wird, modelliert Objekte und deren Eigenschaften, Aktionen und Zustände der Anwendung (siehe Abschnitt 2.7.1). Mit anderen Worten, typisierte Merkmalsstrukturen sind partielle Beschreibungen der Welt, in der das Dialogsystem seine Anwendung findet. Die Methodenspezifikationen erlauben es anzugeben, wie diese partielle Beschreibungen durch den Aufruf von Funktionen in der Anwendung ergänzt werden können.

Definition 8 (Methode). *Sei* Name *die Menge aller Methodennamen. Eine* Methode m *ist durch ein Tupel* $m = \langle name, \langle \tau_1 \circ_1 \pi_1, \ldots, \tau_n \circ_n \pi_n \rangle \rangle$ *gegeben, wobei* $name \in$ Name *der Methodenname ist, die* $\pi_i \in \mathsf{feat}^*$ *Merkmalspfade, die* τ_i *Typen sind, und* $\circ_i$ *für eine der Subsumptionsrelationen* $\sqsupset$ *oder* $\sqsupseteq$ *steht. Die Arität* $arity(n)$ *ist eine totale Funktion, die jedem Methodennamen die Arität der zugehörigen Methode zuordnet. Desweiteren sei* $arg(m, i) = \tau_i$ *eine Funktion, die den Typ des* i*-ten Arguments von* m *angibt, für alle* $1 \leq i \leq n$.

Eine gültige Methodenspezifikation ist beispielsweise

$$\begin{array}{l} display: \\ [name] \quad \sqsupset string, \\ [\text{ADDR}|\text{POSX}\ ] \sqsupset int, \\ [\text{ADDR}|\text{POSY}] \sqsupset int \end{array} \tag{2.1}$$

die verlangt, daß die drei Argumente spezifischer als der Typ *string*, oder *int*, respektive, sind (in anderen Worten, alle Argumente müssen instantiiert sein).

Jedem Methodennamen entspricht eine Funktion in der Anwendung. Methoden werden den Knoten in den Merkmalsstrukturen zugewiesen. Objektorientierte Merkmalsstrukturen (OOMS) sind dann einfache Verallgemeinerungen der Standard-Merkmalsstrukturen wobei die Knoten nicht nur Typinformationen sondern auch Methodeninformation tragen.

Definition 9 (Objekt-Orientierte Merkmalsstrukturen). *Eine* objektorientierte Merkmalsstruktur *über* $\langle \mathsf{Type}, \sqsubseteq \rangle$, Feat *und* Name *ist ein Tupel* $F = \langle Q, \bar{q}, \theta, \delta, \mu \rangle$, *wobei* $Q, \bar{q}, \theta$ *und* δ *definiert sind wie in Definition 2 (i)-(iv) und*

(v) $\mu : Q \rightarrow \mathcal{P}(\mathsf{Meth})$ *eine totale Funktion ist, die jedem Knoten* $q \in Q$ *eine (möglicherweise leere) Menge von Methoden* $\mu(q)$ *so daß jeder Methodenname* $n \in$ Name *in den Methoden in* $\mu(q)$ *höchstens einmal auftritt.*

Sei $\mathcal{OOF}$ *die Menge aller objekt-orientierten Merkmalsstrukturen.*

Die Struktur

$$\left[\begin{array}{ll} obj\_building & \\ \text{NAME} & \text{“}Institut\ für\ Maschinelle\ Sprachverarbeitung\text{“} \\ \text{ADDR} & \left[\begin{array}{ll} addr & \\ \text{STREET} & \text{“}Azenbergstraße\text{“} \\ \text{POSX} & 100 \\ \text{POSY} & 200 \end{array}\right] \\ display() & \end{array}\right] \tag{2.2}$$

ist eine objekt-orientierte Merkmalsstruktur. In diesem Beispiel sind alle Constraints der Methodenspezifikation 2.1 erfüllt.

Wenn q ein Knoten und m eine Methode in $\mu(q)$, dann ist offensichtlich der Zweck der Constraints $\tau_i \circ_i \pi_i$, den Wert der $\delta(q, \pi_i)$ einzuschränken. Die Methode m kann dann aufgerufen werden, wenn alle $arity(m)$ der Methode zugeordneten Constraints erfüllt sind, oder formell, wenn für alle $1 \leq i \leq n$ die folgende Bedingung erfüllt ist:

$\theta(\delta(q, \pi_i)) \sqsupseteq \tau_i$ falls $\circ_i = \sqsupseteq$
$\theta(\delta(q, \pi_i)) \sqsupset \tau_i$ falls $\circ_i = \sqsupset$

Die Tatsache, daß eine Methode m in $\mu(q)$ enthalten ist, bedeutet, daß die der Methode entsprechende Funktion in der Anwendung aufgerufen worden ist.

Die Constraints der Methodenargumente können bezüglich ihrer Einschränkungen partiell geordnet werden. Ein Constraint der Form $\tau_1 \sqsubseteq \pi$ ist offensichtlich nicht strenger als ein Constraint der Form $\tau_2 \sqsubseteq \pi$, wenn $\tau_1 \sqsubseteq \tau_2$ gilt. Außerdem ist ein Constraint der Form $\tau_1 \sqsubset \pi$ strenger als ein Constraint der Form $\tau_1 \sqsubseteq \pi$. Wir können damit auf den Constraints der Methoden eine partielle Ordnung wie folgt definieren.

Definition 10 (Partielle Ordnung der Methodenargumente). *Die partielle Ordnung $\sqsubseteq$ gilt für zwei Methodenargumente $\circ_1\pi_1 : \tau_1$ und $\circ_2\pi_2 : \tau_2$ wenn die folgenden Bedingungen erfüllt sind:*

(i) $\pi_1 = \pi_2$
(ii a) $\circ_1 = \circ_2$ *und* $\tau_1 \sqsubseteq \tau_2$*, oder*
(ii b) $\circ_1 = \sqsubset, \circ_2 = \sqsubseteq$ *und* $\tau_1 \sqsubseteq \tau_2$*.*

Die partielle Ordnung auf den Argumenten kann zu einer partiellen Ordnung auf der Menge der Methoden wie folgt erweitert werden.

Definition 11 (Partielle Ordnung der Methoden). *Seien $m_1 = \langle n, \langle \pi_1 \circ_1 \tau_1, \ldots, \pi_m \circ_m \tau_m \rangle\rangle, m_2 = \langle n, \langle \rho_1 \circ_1 \sigma_1, \ldots, \rho_n \circ_n \sigma_n \rangle\rangle$ zwei Methoden mit demselben Namen n und Arität $arity(n)$. Die Methode m_1 subsumiert die Methode m_2, oder $m_1 \sqsubseteq m_2$ genau dann wenn alle Argumente der Methode m_1 alle Argumente der Methode m_2 subsumieren, oder formell, wenn*

$$\pi_i \circ_i \tau_i \sqsubseteq \rho_i \circ_i \sigma_i$$

für alle $1 \leq i \leq n$ gilt.

Mithilfe der Definition 11 können Subsumption und Unifikation für objekt-orientierte Merkmalsstrukturen definiert werden.

Definition 12 (Subsumption auf OOMS). *Eine objekt-orientierte Merkmalsstruktur $F = \langle Q, \bar{q}, \theta, \delta, \mu \rangle$ subsumiert eine andere objekt-orientierte Merkmalsstruktur $F' = \langle Q', \bar{q}', \theta', \delta', \mu \rangle$ wenn es eine Funktion h gibt, die die Bedingungen (i)-(iii) in Definition 3 erfüllt und es zusätzlich für jeden Knoten $q \in Q$ eine totale Funktion $g : \mu(q) \rightarrow \mu'(h(q))$ gibt so daß für jede Methode $m \in \mu(q)$ die Beziehung $m \sqsubseteq g(m)$ gilt.*

Informell subsumiert eine objekt-orientierte Merkmalsstruktur F eine andere F' genau dann, wenn die zugrundeliegende Standardmerkmalsstruktur von F die F' zugrundeliegende Standardmerkmalsstruktur subsumiert und zusätzlich wenn für jede Methode m in q eine Methode $g(m)$ in $h(q)$ gefunden werden kann, so daß $g(m)$ von m subsumiert wird.

Die Unifikation von objekt-orientierten Merkmalsstrukturen ist ebenso eine einfache Erweiterung der Unifikation von gewöhnlichen Merkmalsstrukturen.

Definition 13 (Unifikation von objekt-orientierten Merkmalsstrukturen). *Seien F, F' zwei objekt-orientierte Merkmalsstrukturen $F \sim \langle Q, \bar{q}, \theta, \delta, \mu\rangle$ und $F' \sim \langle Q', \bar{q}', \theta', \delta', \mu'\rangle$ so daß $Q \cap Q' = \emptyset$. Wir definieren die Äquivalenzrelation $\bowtie$ auf $Q \cup Q'$, $\theta^{\bowtie}$ und $\delta^{\bowtie}$ wie in Definition 4. Die Äquivalenzrelation $\sim$ auf $(\mu \cup \mu')([q]_{\bowtie})$ ist definiert als die kleinste Äquivalenzrelation so daß*

$$m \sim m' :\Leftrightarrow name(m) = name(m')$$

Die Unifikation von F und F' ist dann definiert als

$$F \sqcup F' = \langle (Q \cup Q')/_{\bowtie}, [\bar{q}]_{\bowtie}, \theta^{\bowtie}, \delta^{\bowtie}, \mu^{\bowtie}\rangle$$

wobei $\theta^{\bowtie}$ und $\delta^{\bowtie}$ definiert sind wie in Definition 4, und

$$\mu^{\bowtie}([q]_{\bowtie}) = \bigcup_{m \in (\mu \cup \mu')([q]_{\bowtie})} \left\{ \bigsqcup_{m' \sim m} [m']_{\sim} \right\}$$

Die Unifikation zweier objekt-orientierter Merkmalsstrukturen ist undefiniert, wenn die Unifikation der zugrundeliegenden typisierten Merkmalsstrukturen oder $\mu^{\bowtie}([q]_{\bowtie})$ undefiniert ist.

Die obige Definition erweitert die Unifikation von gewöhnlichen Merkmalsstrukturen zu objekt-orientierten Merkmalsstrukturen. Um die Methodenfunktion μ zu konstruieren, werden alle Knoten in einer Äquivalenzklasse betrachtet. Zwei Methoden sind äquivalent bezüglich $\sim$, wenn sie für Knoten in derselben Äquivalenzklasse definiert sind und denselben Namen haben. Die resultierende Methodenmenge für die Knotenäquivalenzklasse wird konstruiert, indem für jede Methodenäquivalenzklasse die kleinste obere Schranke bestimmt wird.

2.5.2 Wohltypisierte objekt-orientierte Merkmalsstrukturen

Genau wie nicht jedes Merkmal jedem Typ zugeordnet werden kann, kann auch nicht jede Methode jedem Typ zugeordnet werden. Im obigem Beispiel ist die *display* Methode nur für Instanzen solcher Typen sinnvoll, die auch auf der Karte angezeigt werden können.; zum Beispiel *obj_building*. Deswegen werden die Wohlgeformtheitsspezifikationen für Methoden erweitert. Dies erfolgt, in dem die Funktion *Approp* um eine weitere partielle Funktion $Method$ ergänzt wird. Zur Erinnerung: *Approp* bildet einen Typen τ und ein Merkmal f auf den Typen ρ ab, den der Wert f, wenn angewandt auf τ, haben soll. Die partielle Funktion $Method$ hingegen bildet einen Typen und einen Methodennamen auf die Typen der Argumente der Methode ab. Wenn wir die Spezifikation

$$Method(obj\_building, display) = \langle string, int, int\rangle$$

annehmen, erfüllt das obige Beispiel die Wohlgeformtheitsspezifikationen. Wir können zusätzlich noch die Argument-Constraints verschärfen, sollte dies nötig sein. Dies geschieht in Analogie zu der Monotonie-Eigenschaft der Wohlgeformtheitsspezifikationen.

Definition 14 (Methoden-Wohlgeformtheits Spezifikation). *Eine* Methoden-Wohlgeformtheits Spezifikation *über einer Typenhierarchie* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *und einer Menge von Methodennamen* Name *ist eine partielle Funktion* $Method : \mathsf{Type} \times \mathsf{Name} \to \bigcup_{n>0} \mathsf{Type}^n$ *so daß die folgenden Bedingungen erfüllt sind:*

(i) **Methoden-Einführung**
für jede Methodennamen $n \in \mathsf{Name}$ *gibt es einen allgemeinsten Typ* $Intro(n) \in \mathsf{Type}$ *so daß* $Method(Intro(m), m)$ *wohldefiniert ist,*

(ii) **Monotone Kovariante Vererbung von Methoden**
wenn $Method(\sigma, n)$ *wohldefiniert ist und* $\sigma \sqsubseteq \rho$ *gilt, dann ist auch* $Method(\rho, n)$ *wohldefiniert und es gilt außerdem* $Method(\sigma, n) \sqsubseteq Method(\rho, n)$.

(iii) **Konstante Stelligkeit**
$Method(\sigma, n) \in \mathsf{Type}^{arity(n)}$ *für alle Paare* σ, m *für welche* $Method(\sigma, m)$ *definiert ist, und*

(iv) **Wohltypisierte Argument-Constraints**
für alle $m \in \mu(q)$ *gilt, daß* $Approp(intro(m), \pi_i)$ *wohldefiniert ist und so daß* $Approp(intro(m), \pi_i) \sqcup \tau_i$ *ebenfalls wohldefiniert ist*

Bedingung (i) stellt sicher, daß es – genau wie bei den Merkmalen – einen allgemeinsten Typen gibt, der die Methode einführt und von dem all Typen, auf die die Methode angewandt werden soll, erben. Bedingung (ii) stellt sicher, daß, wenn eine Methode spezialisiert wird, dies monoton geschieht. Diese Bedingung entspricht der Monotonie/Aufwärts-Abgeschlossenheit der Wohlgeformtheitsspezifikation. Bedingung (iii) hat keine Entsprechung in Definition 5 und stellt lediglich sicher, daß die Stelligkeit einer Methode konstant bleibt, wenn sie spezialisiert wird. Bedingung (iv) stellt sicher, daß die Argumentconstraints, wenn sie in die Merkmalsstruktur eingebracht werden, Wohltypisierbarkeit der zugrundeliegenden Standardmerkmalsstruktur erhalten.

Mit der obigen Definition kann nun bestimmt werden, was es für eine objektorientierte Merkmalsstruktur bedeutet, wohltypisiert zu sein. Es ist der Sinn dieser Definition sicherzustellen, daß Methoden nur an den für sie erlaubten Knoten auftreten und alle Methoden, deren Argument-Constraints erfüllt sind, auch in der Merkmalsstruktur vorhanden sind.

Definition 15 (Wohltypisierte objekt-orientierte Merkmalsstrukturen). *Eine objekt-orientierte Merkmalsstruktur* $F = \langle Q, \bar{q}, \theta, \delta, \mu \rangle$ *heißt* wohltypisiert*, wenn die zugrundeliegende Standardmerkmalsstruktur* $\langle Q, \bar{q}, \theta, \delta \rangle$ *wohltypisiert ist und desweiteren folgende Bedingungen gelten:*

(i) **genau die Methoden mit erfüllten Constraints sind Teil der Merkmalsstruktur**
eine Methode m *ist in* $\mu(q)$ *genau dann, wenn* $intro(m) \sqsubseteq \theta(q)$ *und* $\tau_i \circ_i \theta(\delta(q, \pi_i))$ *gelten, und*

2.5.3 Methodenaufruf

Die Methodenspezifikation sind so definiert worden, daß sie als eine direkte Generalisierung der Merkmalsspezifikationen aufgefaßt werden können. Insbesondere entsprechen die Eigenschaften (i) und (ii) (Methodeneinführung und kovariante Vererbung) genau den Eigenschaften (i) und (ii) der Merkmalsspezifikationen. Deswegen überrascht es nicht, daß mit einer Inferenzprozedur die Methoden in die Merkmalsstrukturen eingebracht werden können.

Theorem 1 (Inferenzprozedur zum Methodenaufruf). *Es gibt eine partielle Funktion $OOInf : \mathcal{OOF} \to \mathcal{OOF}$ die jeder objekt-orientierten Merkmalsstruktur die kleinste wohltypisierte objekt-orientierte Merkmalsstruktur zuordnet, falls eine solche existiert. Anderenfalls ist $OOInf$ undefiniert.*

Beweis.
Sei $F = \langle Q, \bar{q}, \theta, \delta, \mu \rangle$ eine objekt-orientierte Merkmalsstruktur. Ist F wohltypisiert, wird die Prozedur verlassen. Ist F nicht wohltypisiert, wird zunächst die Typinferenz für typisierte Merkmalsstrukturen auf die F zugrundeliegende Merkmalsstruktur $\langle Q, \bar{q}, \theta, \delta \rangle$angewandt. Wir betrachten nun die Bedingung in Definition 15.

- $\Rightarrow$ Sei $m \in \mu(q)$ eine Methode, so daß die rechte Seite von Bedingung (i) verletzt ist. Wenn $intro(m) \not\sqsubseteq \theta(q)$ gilt, dann setzen wir $\theta(q) \leftarrow intro(m) \sqcup \theta(q)$. Existiert $intro(m) \sqcup \theta(q)$ nicht, ist F nicht wohltypisierbar. Wenn ein Argument-Constraint $\tau_i \circ_i \pi_i$ verletzt ist, setzen wir $\theta(\delta(q, \pi_i)) \leftarrow \theta(\delta(q, \pi_i)) \sqcup \tau_i$. Existiert $theta(\delta(q, \pi_i)) \sqcup \tau_i$ nicht, ist F nicht wohltypisierbar.
- $\Leftarrow$ Sei nun $m \notin \mu(q)$ eine Methode, für die die rechte Seite von Bedingung (i) gilt, dann setzen wir $\mu(q) \leftarrow \mu(q) \cup \{m\}$.

Wie wenden nun abwechselnd die Standard Inferenzprozedur und die hier angegebene Prozedur auf F solange an, bis ein Fixpunkt erreicht wird, oder eine der beiden Prozeduren feststellt, daß die Struktur nicht wohltypisiert werden kann. In diesem Fall ist $OOInf$ undefiniert.

Die Standardinferenzprozedur endet in endlich vielen Schritten. Da es nur endlich viele Methodenspezifikationen gibt, können in den obigen Schritten nur endlich viele Methodennamen zu der Merkmalsstruktur hinzugefügt und nur endlich viele Argumentconstraints eingebracht werden. Damit endet die angegebene Prozedur in endlich vielen Schritten.

Es bleibt zu zeigen, daß $OOInf(F)$ wohltypisiert ist, falls $OOInf(F)$ wohldefiniert ist. Wir nehmen an, daß $OOInf$ wohldefiniert, aber nicht wohltypisiert ist. Dann ist entweder die zugrundeliegende Standardmerkmalsstruktur nichtwohltypisiert, oder aber die Bedingung in Definition 15 ist verletzt. Im ersten Fall gäbe es dann noch einen Schritt, den die Standardinferenzprozedur auszuführen hat. was ein Widerspruch zu der Annahme ist, daß $OOInf$ terminiert hat. Im zweiten Fall gilt das gleiche Argument. ■

Die Merkmalsstruktur

$$\left[\begin{array}{ll} \textit{obj\_building} & \\ \textsc{name} & \textit{“Institut für Maschinelle Sprachverarbeitung“} \\ \textsc{addr} & \left[\begin{array}{ll} \textit{addr} & \\ \textsc{street} & \textit{“Azenbergstraße“} \\ \textsc{posx} & \textit{100} \\ \textsc{posy} & \textit{200} \end{array}\right] \end{array}\right] \tag{2.3}$$

ist nicht wohltypisiert, da alle Argument-Constraints der Methode *display*() erfüllt sind, und weiterhin *display*() ein für *obj_building* erlaubter Methodenname ist, aber der Methodenname nicht in der Merkmalsstruktur notiert ist. Die im Beweis angegebene Inferenzprozedur, wenn angewandt auf diese Merkmalsstruktur, stellt fest, daß die Argument-Constraints der Methode *display*() (wie in (2.1) erfüllt sind. Deswegen wird der Methodenname zu der Merkmalsstruktur hinzugefügtund wir erhalten die Struktur in (2.2) als Ergebnis. Gleichzeitig wird beim Hinzufügen des Methodennamens die Prozedur *display* mit den Argumenten *“Institut für Maschinelle Sprachverarbeitung“*, 100 und 200 aufgerufen.

Eine Methodenspezifikation implementiert nicht ein bestimmtes Verhalten. Die Inferenzprozedur stellt lediglich fest, ob die Merkmalsstruktur genügend Information enthält, sodaß die der Methode zugeordnete Funktion aufgerufen werden kann. Wenn das der Fall ist, wird die Funktion mit den in der Merkmalsstruktur enthaltenen Argumenten aufgerufen und der Aufruf der Funktion dadurch vermerkt, daß der Methodenname in die Methodenmenge des entsprechenden Knotens aufgenommen wird. Diese Vorgehensweise ermöglicht eine vollständige Entkopplung der Spezifikation des Methodenaufrufes und der tatsächlichen Ausführung des Aufrufes. Natürlich kann man in aufgabenorientierten Dialogsystem von dieser Eigenschaft gut Gebrauch machen, da man so spezifizieren, wann genügend Information vorhanden ist, um eine Funktion aufzurufen, ohne sagen zu müssen, was diese Funktion tut.

Auch in anderer Hinsicht bietet dieser Ansatz Vorteile. Carpenter [1992] betont, daß typisierte Merkmalsstrukturen *partielle Beschreibungen* von Sachverhalten sind. Da die Argumentconstraints der Methoden bestimmen, wann genügend Information vorhanden ist, um eine Methode aufzurufen, können auch Methoden auf partielle Beschreibungen angewandt werden, solange nur die Argumentconstraints erfüllt sind. Dies ist in Kontrast zu beispielsweise objekt-orientierten Programmiersprachen, wo Objekte vollständig konstruiert sein müssen (d.h. vollständige Information über die modellierten Objekte muß vorliegen), bevor Methoden auf die Objekte angewandt werden können.

2.6 Multidimensionale Merkmalsstrukturen

Dieser Abschnitt beschäftigt sich mit einer Erweiterung typisierter Merkmalsstrukturen, bei der neben der semantischen Typinformation weitere für den Dialogfluß relevante Information in den Knoten repräsentiert werden kann. Diese Information kann zum Beispiel Konfidenzinformation des Spracherkenners

enthalten, oder die Anzahl Prompts, mit der ein Wert erfragt worden ist. Dialogstrategien können dann nicht nur in Abhängigkeit der semantischen Information, sondern auch in Abhängigkeit dieser Meta-Information Entscheidungen treffen.

Als Erinnerung sei erwähnt, daß die Standardsubsumptions und -unifikationsprozeduren für typisierte Merkmalsstrukturen lediglich voraussetzen, daß die Typen in einer unteren Halbordnung angeordnet. Die Generalisierung besteht nun darin, die Typfunktion θ in den Merkmalsstrukturen durch eine Typvektorfunktion zu ersetzen, sodaß jedem Knoten nicht nur ein Typ, sondern ein n dimensionaler Vektor $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$ von Elementen zugeordnet wird. Die Elemente v_i stammen ihrerseits aus einer unteren Halbordnung $\langle V_i, \sqsubseteq_i \rangle$. Eine solche Halbordnung kann natürlich durch die Typenhierarchie gegeben sein. Weitere denkbare Halbordnungen repräsentieren den Eingabekanal, Konfidenzintervalle und andere mehr.

Multidimensionale Merkmalsstrukturen können dann genauso unifiziert und auf Subsumption getestet werden wie Standardmerkmalsstrukturen auch. Die Abbbildung 2.5 zeigt ein Beispiel einer Standard- und einer Multidimensionalen Merkmalsstruktur.

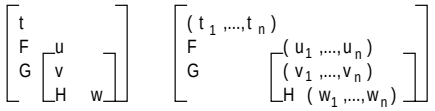


Abbildung 2.5. Eine typisierte und eine multidimensionale Merkmalsstruktur.

Die Definitionen 3 und 4 für Subsumption und Unifikation von typisierten Merkmalsstrukturen erfordern lediglich, daß die Typinformation der Knoten in einer unteren Halbordnung geordnet sind. In multidimensionalen Merkmalsstrukturen ist diese Halbordnung durch die Ordnung der Vektoren gegeben wie folgt:

$$\mathbf{v} \sqsubseteq \mathbf{w} :\Leftrightarrow v_i \sqsubseteq_i w_i \qquad \forall 1 \leq i \leq n$$

Dies bedeutet, daß Unifikation und Subsumption von typisierten Merkmalsstrukturen direkt für multidimensionale Merkmalsstrukturen übernommen werden kann.

2.7 Anwendungen in der Dialogverarbeitung

In diesem Abschnitt wird dargestellt, wie die in diesem Kapitel eingeführten Konzepte für die Verarbeitung natürlichsprachlichen Dialogs genutzt werden können.

2.7.1 Typenhierarchie als Objektmodell

Die Typenhierarchie wird verwendet, um die semantischen Konzepte der zu modellierenden Domäne miteinander in Beziehung zu setzen. Damit ist es möglich, Vererbungsbeziehungen IS-A und Teilbeziehungen IS-PART-OF zwischen den in

der Domäne verwendeten Aktionen, Objekten und deren Eigenschaften auszudrücken.

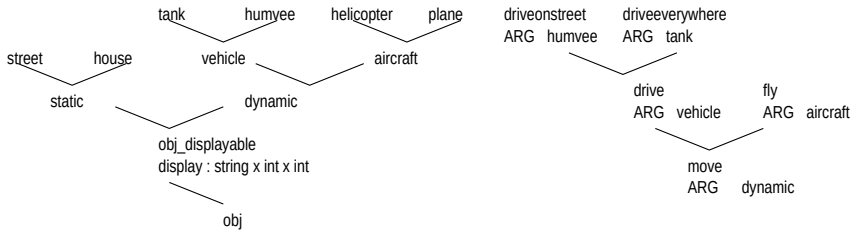


Abbildung 2.6. Ein Auszug aus einem objekt-orientierten Domänenmodell für eine militärische Anwendung.

2.7.2 Multidimensionale Merkmalsstrukturen

Anzahl der Frageversuche. Um Mißkommunikationen adäquat behandeln zu können, muß das Dialogsystem repräsentieren können, wie oft der Wert für einen gegebenen Merkmalspfad erfragt worden ist. Um zu verhindern, daß der Dialog zu einer nicht endenden Schleife degradiert, wird die Dialogstrategie gewechselt, wenn ein gewisser Schwellwert überschritten wird.

Eingabemedium. Zusätzlich zu der Anzahl der Frageversuche kann man für jeden Merkmalswert die Eingabemodalität mit abspeichern. Dadurch kann die Dialogstrategie Anpassungen bei den Klärungsfragen vornehmen und beispielsweise vorschlagen, eine andere Eingabemodalität zu verwenden, sollte die Qualität des Dialogs sich verschlechtern.

Um die Inferenzmechanismen monoton zu halten, wird nicht repräsentiert, daß eine gegebene Eingabemodalität noch nicht verwendet wurde. Anstatt dessen werden alle Eingabemodalitäten repräsentiert, die zur Erbringung der Typinformation bisher benutzt worden sind. Damit werden bei Verwendung zusätzlicher Eingabemodalitäten spezifischere Elemente in den Vektoren gespeichert, und die Verarbeitung bleibt somit monoton. Die partielle Ordnung ist in Abbildung 2.7 (a) gezeigt.

Konfidenzmaße. In ähnlicher Weise können Konfidenzannotationen des Spracherkenners in multidimensionalen Merkmalsstrukturen repräsentiert werden. Die partielle Ordnung der Konfidenzannotation ist in Abbildung 2.7 (b) gezeigt.

2.8 Diskussion

Das Ziel der in diesem Kapitel eingeführten Erweiterungen ist es, das Dialogsystem mit ausdrucksstarken Repräsentationen zu versehen, deren Informationsgehalt die Interaktion mit der Anwendung und mit dem Benutzer steuert.

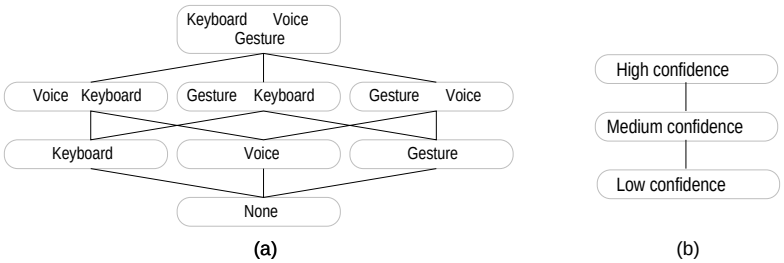


Abbildung 2.7. (a) Partielle Ordnung, die die Eingabemodalitäten repräsentiert. (b) Partielle Ordnung zur Repräsentation der Konfidenzannotation.

Die objekt-orientierten Erweiterungen ermöglichen es, immer dann Dienste der Anwendung aufzurufen, wenn eine gewisse Menge an Information im Diskurs etabliert worden ist. Der Dienst der Anwendung kann beispielsweise Objekte immer dann anzeigen, sobald ihre Position auf dem Bildschirm bekannt ist (durch eine Datenbankanfrage) oder in einem programmierbaren Videorecorder die Länge des aufzunehmenden Filmes berechnen.

Die Multidimensionalen Merkmalsstrukturen erlauben die Repräsentation von zusätzlicher Information, die für die Dialogverarbeitung von Bedeutung ist. Dies schließt Konfidenzannotationen und Eingabemodalität mit ein.

Der entscheidende Punkt bei den hier vorgestellten Erweiterungen von typisierten Merkmalsstrukturen ist, daß sie die Prinzipien der typisierten Merkmalsstrukturen erhalten, nämlich das informationsbasierte, oder datengetriebene, Verarbeitung von partieller Information. Damit ist es möglich, Bedingungen an den Dialogzustand durch Merkmalsstrukturen zu formulieren.

2.9 Zusammenfassung

In diesem Kapitel wurden typisierte Merkmalsstrukturen präsentiert. Desweiteren wurden zwei Erweiterungen von typisierten Merkmalsstrukturen vorgeschlagen. Die eine Erweiterung, objektorientierte Merkmalsstrukturen, ermöglichen die Dienstanforderung von Diensten der Anwendung in Abhängigkeit der im Diskurs repräsentierten Information. Die andere Erweiterung, multidimensionale Merkmalsstrukturen, erlaubt die Repräsentation von für den Ablauf des Dialogs wichtiger Zusatzinformation. Beide Erweiterungen können in domänen- und zielsprachunabhängiger Art und Weise eingesetzt werden.

Kapitel 3

Ambiguität und Generalisierungen von Merkmalsstrukturen

Ein Dialogsystem muß interaktiv solange Informationen vom Anwender erfragen, bis der vom Anwender angeforderte Dienst eindeutig bestimmt und für die Dienstausführung benötigten Parameter bekannt sind. Ein einfaches Verfahren, dies zu erreichen, ist das folgende: erst werden alle verfügbaren Dienste aufgezählt und der Anwender gebeten, einen zu wählen (etwa *"Would you like to get directions or make a hotel reservation?"*). Anschließend, nachdem der Anwender einen Dienst ausgewählt hat, werden die Werte für alle Parameter erfragt (etwa *"Which type of place would you like to go?"* und so weiter).

Dieser Ansatz hat mehrere Probleme. Zum einen hat der Anwender wenig Kontrolle über den Dialog; die Interaktion ist vom Dialogsystem gesteuert und nicht sehr natürlich. Desweiteren kann Information aus anderen Wissensquellen nur sehr schwer in den Dialog integriert werden.

Aus diesem Grunde wurde in der vorliegenden Arbeit ein *informationsbasierter* oder *datengetriebener* Ansatz verfolgt. Hierbei wird versucht, so früh wie möglich einen Datenbankzugriff durchzuführen. Anschließend werden dem Anwender Disambiguierungsfragen *in Abhängigkeit der Ergebnismenge* der Datenbankanfrage dargeboten. Das heißt, wenn die erste Äußerung des Anwenders etwa *"Give me directions to a restaurant in Squirrel Hill"* ist, und sich die in der Ergebnismenge repräsentierten Objekte nicht in der Preiskategorie, aber vielleicht in der Nationalität unterscheiden, braucht nach Preiskategorie nicht gefragt zu werden. Im Gegensatz dazu unterscheiden sich die repräsentierten Objekte in der Nationalität, womit dieses Merkmal Grundlage für Disambiguierung sein kann.

Darüber hinaus ist für einen informationsbasierten Ansatz die Quelle der Information irrelevant. Information kann somit auch aus Benutzerprofilen oder ähnlichen Wissensquellen beigetragen werden, um die Länge des Dialogs zu verkürzen.

In diesem Kapitel werden daher Repräsentationen eingeführt, die es ermöglichen, Gemeinsamkeiten und Unterschiede von Objektbeschreibungen effizient

zu repräsentieren. Auf Basis dieser Repräsentationen wird dann der semantische Inhalt für Disambiguierungsfragen bestimmt.

3.1 Einführung

Die in dieser Arbeit verwendeten Repräsentationen sind die im vorigen Kapitel beschriebenen typisierten Merkmalsstrukturen. Gegeben eine Menge von Merkmalsstrukturen $\{F_1, \ldots, F_n\}$ über einer Typenhierarchie $\langle \mathsf{Type}, \sqsubseteq \rangle$, ist es recht einfach, die Unterschiede[1] zu bestimmen: Man betrachtet einen Merkmalspfad π, der in zumindest zwei Merkmalsstrukturen F_i und F_j, $i \neq j$, definiert ist, und vergleicht den Typ τ des Wertes von $F_i@\pi$ mit dem Typ σ des Wertes von $F_j@\pi$. Sind τ und σ unterschiedlich,[2] wissen wir, daß zwar die Werte von π in F_i und F_j unterschiedlich sind. Sie haben jedoch die Information gemeinsam, daß für den Wert von π in F_i und F_j der Typ $\tau \sqcap \sigma$ gilt. Diese Information kann repräsentiert werden als

$$\pi : \rho^{(i,j)} \begin{cases} \tau^{(i)} \\ \sigma^{(j)} \end{cases} \tag{3.1}$$

wobei $\rho = \tau \sqcap \sigma$ gilt. Die Repräsentation (3.1) ist zu lesen wie folgt. Wenn ein Typ τ mit einem Index i annotiert ist, ist die durch τ repräsentierte Information in der Merkmalsstruktur F_i vorhanden. Die Aussage von (3.1) ist also, daß die Merkmalsstrukturen an der Stelle π die Information ρ (aber nicht mehr) gemeinsam haben. Desweiteren ist die Aussage, daß in F_i an der Stelle π die Information τ, und in F_j an derselben Stelle die Information σ gilt.

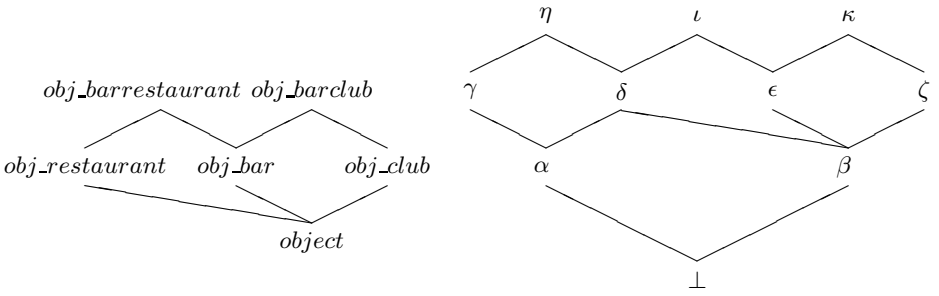
Offensichtlich gilt in (3.1), daß $\rho \sqsubseteq \tau$ und $\rho \sqsubseteq \sigma$. Deswegen kann die durch (3.1) beschriebene partielle Ordnung in die Typenhierarchie $\langle \mathsf{Type}, \sqsubseteq \rangle$ eingebettet werden. Allgemein gilt also: wenn $\tau_1, \ldots, \tau_n$ die Typen der Werte an den Stellen $F_i@\pi$ sind, dann lassen sich die Gemeinsamkeiten und Unterschiede der Typen in der partiellen Ordnung repräsentieren, die in $\langle \mathsf{Type}, \sqsubseteq \rangle$ eingebettet werden kann.

Die in diesem Kapitel vorgeschlagene Generalisierung der typisierten Merkmalsstrukturen ist dann offensichtlich. Eine generalisierte Merkmalsstruktur ist eine Merkmalsstruktur, in der die Knoten $q, \ldots$ durch generalisierte Knoten $Q, \ldots$ ersetzt werden, wobei jeder generalisierte Knoten Q die oben hergeleitete, zu $\langle \mathsf{Type}, \sqsubseteq \rangle$ homomorphe Struktur aufweist. Die Repräsentation in (3.1) ist ein Beispiel eines solchen generalisierten Knotens.

In (3.1) wird die Information repräsentiert, die F_i und F_j gemeinsam ist. Analog dazu werden in diesem Kapitel auch Repräsentationen entwickelt, die die Information in F_i und F_j repräsentiert werden, die zueinander kompatibel ist. Damit lassen partielle Unifikationen repräsentieren.

[1] Wir betrachten im Moment nur Unterschiede in der Typinformation und lassen Pfadgleichungen außer acht.

[2] In diesem Beispiel nehmen wir der Darstellung wegen an, daß $\tau \not\sqsubseteq \sigma$ und $\sigma \not\sqsubseteq \tau$ gilt. Dies ist jedoch für des Anwendung des entwickelten Verfahrens keine Voraussetzung.



(a) (b)

Abbildung 3.1. Die in den Beispielen verwendete Typenhierarchien. Abbildung (a) zeigt eine einfachere Typenhierarchie, in der konkrete Objekte repräsentiert werden. Abbildung (b) zeigt eine abstrakte Typenhierarchie.

3.2 Generalisierte Merkmalsstrukturen

In diesem Abschnitt werden die in der Einführung skizzierten Generalisierungen konkretisiert. Die Generalisierung besteht darin, daß die Knoten typisierter Merkmalsstrukturen durch generalisierte Knoten ersetzt werden. Damit können mehrere Merkmalsstrukturen $F_1, \ldots F_n$ in einer generalisierten Merkmalsstruktur repräsentiert werden. Zudem ist aus G auch die Information ersichtlich, die jeder Teilmenge von $F_1, \ldots F_n$ gemeinsam ist (alternativ, die mit jeder Teilmenge von $F_1, \ldots F_n$ kompatibel ist).

3.2.1 Generalisierte Knoten

Wir betrachten zunächst generalisierte Knoten. Um die gemeinsame (kompatible) Information in den Knotenmengen ablesen zu können, wird jeder generalisierte Knoten mit einer Menge von Typen versehen. Jeder Typ τ verfügt zusätzlich über eine Indexmenge, die aussagt, welche der n Merkmalsstrukturen die Information τ enthält (mit der Information τ kompatibel ist).

Zur Illustration der generalisierten Merkmalsstrukturen betrachten wir zunächst atomare Merkmalsstrukturen. Seien F_i, F_j und F_k atomare Merkmalsstrukturen der Typen *obj_restaurant*, *obj_bar* und *obj_club* (die Typen entstammen der Typenhierarchie in Abbildung 3.1; zunächst werden die anschaulichen Typen der linken Typenhierarchie verwendet, später werden der Übersichtlichkeit halber die griechischen Symbole verwendet). Wir betrachten nun die generalisierte Merkmalsstruktur, die die gemeinsame Information von F_i, F_j und F_k repräsentiert. Die Index-Mengen vermerken die Indices der Merkmalsstrukturen, die die mit der Index-Menge versehene Information enthält. Die Knoten der atomaren Merkmalsstrukturen sind durch ausgefüllte Punkte dargestellt und mit dem Index der Merkmalsstruktur annotiert, zu der sie gehören.

$$\left[object^{i,j,k} \left\{ \begin{array}{l} \bullet^i \; obj\_restaurant^i \\ \bullet^j \; obj\_bar^j \\ \bullet^k \; obj\_club^k \end{array} \right. \right] \tag{3.2}$$

Die Tatsache, daß $object^{i,j,k}$ in der Repräsentation enthalten ist, sagt aus, daß alle drei Merkmalsstrukturen i, j, k die Information *object* enthalten. Die

Tatsache, daß es in der Repräsentation keinen weiteren Typ mit zwei oder mehr Indices gibt, sagt aus, daß *object* die einzige Information ist, die sich zwei der Merkmalsstrukturen teilen. Würde eine vierte atomare Merkmalsstruktur $F_l = [obj\_barclub]$ zu 3.2 hinzugefügt, sähe die resultierende Repräsentation wie folgt aus.

$$\left[object^{i,j,k,l} \left\{ \begin{array}{l} \bullet^i \ obj\_restaurant^i \\ \bullet^j \ obj\_bar^{j,l} \\ \bullet^k \ obj\_club^{k,l} \end{array} \right\} \bullet^l \ obj\_barclub^l \right] \tag{3.3}$$

Wir konstruieren nun die partiell unifizierte Merkmalsstruktur von F_i, F_j und F_k. Da für obj_bar, obj_club und $obj\_restaurant$ keine gemeinsame obere Schranke existiert, werden maximal kompatible Teilmengen von obj_bar, obj_club und $obj\_restaurant$ gebildet und deren obere Schranken abgespeichert. Die generalisierte atomare Merkmalsstruktur von F_i, F_j und F_k ist dann gegeben durch die Repräsentation in

$$\left[object^{i,j,k} \left\{ \begin{array}{ll} \bullet^i \ obj\_restaurant^{i,j} & \left. \begin{array}{l} \\ \\ \end{array} \right\} obj\_barrestaurant^{i,j} \\ \bullet^j \ obj\_bar^{i,j,k} & \\ \bullet^k \ obj\_club^{j,k} & \left. \begin{array}{l} \\ \\ \end{array} \right\} obj\_barclub^{j,k} \end{array} \right] \tag{3.4}$$

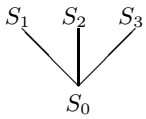
Die Aussage von der Repräsentation in 3.4 ist, daß alle drei Merkmalsstrukturen kompatibel mit obj_bar sind, wie an der Index-Menge von obj_bar zu sehen ist. Darüber hinaus wird ausgesagt, daß $obj\_barrestaurant$ und $obj\_barclub$ lediglich mit den Merkmalsstrukturen F_i und F_j, und F_j und F_k, respektive, kompatibel sind. Würde die vierte atomare Merkmalsstruktur $F_l = [obj\_barclub]$ zu 3.4 hinzugefügt, sähe die resultierende Repräsentation wie folgt aus.

$$\left[object^{i,j,k,l} \left\{ \begin{array}{ll} \bullet^i \ obj\_restaurant^{i,j} & \left. \begin{array}{l} \\ \\ \end{array} \right\} \quad obj\_barrestaurant^{i,j} \\ \bullet^j \ obj\_bar^{i,j,k,l} & \\ \bullet^k \ obj\_club^{j,k,l} & \left. \begin{array}{l} \\ \\ \end{array} \right\} \bullet^l \ obj\_barclub^{j,k,l} \end{array} \right] \tag{3.5}$$

In beiden Fällen wird Information nach unterschiedlichen Kriterien separiert. Im ersten Fall ist es gemeinsame Information, im zweiten Fall ist es kompatible Information. Es sind zwar die Interpretation von 3.4 und 3.2 unterschiedlich, aber die zugrundeliegenden Datenstrukturen sind dieselben. Aus diesem Grunde werden alle folgenden Definitionen und Algorithmen für den allgemeinen Fall angegeben.

Die Datenstrukturen für die Repräsentationen 3.2 und 3.4 sind gegeben in Abbildung 3.2 und 3.3. In den obigen Darstellungen sind die Knoten der ursprünglichen Merkmalsstrukturen durch ausgefüllte Punkte dargestellt. Dies entspricht den Knotenmengen S in den Abbildungen 3.2 und 3.3. Die Knotenmengen in der obigen Definition nehmen die Knoten der Merkmalsstrukturen auf, die in der generalisierten Merkmalsstruktur gespeichert sind. Die partielle Ordnung der Knotenmengen, induziert durch die Typen, ist repräsentiert durch eine binäre Relation $\sigma \subseteq S \times S$. Jeder Knotenmenge S ist durch eine Typfunktion θ ein Typ zugeordnet. Wir verlangen, daß ein Homomorphismus existiert zwischen der Menge der Knotenmenge und ihren Typen, so daß $\sigma(S, S')$ genau

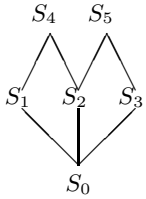
$$\left[object^{i,j,k} \left\{ \begin{array}{l} \bullet^i \ obj\_restaurant^i \\ \bullet^j \ obj\_bar^j \\ \bullet^k \ obj\_club^k \end{array} \right. \right]$$



	θ	i	Knoten
S_0	$object$	$\{i,j,k\}$	$\emptyset$
S_1	$obj\_restaurant$	$\{i\}$	q_i
S_2	obj_bar	$\{j\}$	q_j
S_3	obj_club	$\{k\}$	q_k

Abbildung 3.2. Strukturrelation, Typ- und Indexfunktion für die in 3.2 gezeigten unterspezifizierten Knoten.

$$\left[object^{i,j,k} \left\{ \begin{array}{l} \bullet^i \ obj\_restaurant^{i,j} \\ \bullet^j \ obj\_bar^{i,j,k} \\ \bullet^k \ obj\_club^{j,k} \end{array} \right. \begin{array}{l} \left. \begin{array}{l} \\ \\ \end{array} \right\} obj\_barrestaurant^{i,j} \\ \left. \begin{array}{l} \\ \\ \end{array} \right\} obj\_barclub^{j,k} \end{array} \right]$$



	θ	i	Knoten
S_0	$object$	$\{i,j,k\}$	$\emptyset$
S_1	$obj\_restaurant$	$\{i,j\}$	q_i
S_2	obj_bar	$\{j,j,k\}$	q_j
S_3	obj_club	$\{j,k\}$	q_k
S_4	$obj\_barrestaurant$	$\{j,j\}$	$\emptyset$
S_5	$obj\_barclub$	$\{j,k\}$	$\emptyset$

Abbildung 3.3. Strukturrelation, Typ- und Indexfunktion für den in 3.4 gezeigten partiell unifizierten Knoten.

dann, wenn $\theta(S) \sqsubseteq \theta(S')$. Diese Überlegungen führen dann zu der folgenden Definition von generalisierten Knoten.

Definition 1 (Generalisierte Knoten). *Ein* generalisierter Knoten Q *über einer Typenhierarchie* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *ist ein Tupel* $Q = \langle \mathcal{S}, \theta, \sigma, i \rangle$, *wobei*

1 $\mathcal{S} = \{S_1, \ldots, S_n\}$ *eine Menge möglicherweise leerer Knotenmengen* $S_k = \{q_1, \ldots, q_{n_k}\}$ *ist so daß* $S_i \cap S_j = \emptyset$ *für alle* $i \neq j$,
2 $\theta : \mathcal{S} \rightarrow \mathsf{Type}$ *eine vollständig definierte Typfunktion ist,*
3 $\sigma \subseteq \mathcal{S} \times \mathcal{S}$ *eine untere Halbordnung, die in* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *eingebettet werden kann, ist,*
4 $i : \bigcup_{S \in \mathcal{S}} S \rightarrow \mathcal{P}(\mathcal{I})$ *eine vollständig definierte Indexfunktion ist,*

so daß, wenn immer $\sigma(S, S')$ *gilt, auch* $\theta(S) \sqsubseteq \theta(S')$ *gilt; außerdem wird verlangt, daß*

$$i(S) = \{i(q) : q \in S\} \cup \bigcup_{S' : \sigma(S,S')} i(S') \tag{3.6}$$

Die Knotenmengen in der obigen Definition nehmen die Knoten der Merkmalsstrukturen auf, die in der generalisierten Merkmalsstruktur gespeichert sind.

3.2.2 Generalisierte Merkmalsstrukturen

Die Verallgemeinerung von Merkmalsstrukturen zu generalisierten Merkmalsstrukturen ist mithilfe von Definition 1 einfach: eine generalisierte Merkmalsstruktur ist eine Merkmalsstruktur, deren Knoten generalisierte Knoten sind. Desweiteren verfügen generalisierte Merkmalsstrukturen über eine Indexfunktion und eine Strukturrelation, die jeweils die Vereinigung der Indexfunktionen und der Strukturrelationen der Knoten in der generalisierten Merkmalsstruktur sind.

Definition 2 (Generalisierte Merkmalsstruktur). *Eine* Generalisierte Merkmalsstruktur *ist ein Tupel* $G = \langle \mathcal{Q}, \bar{Q}, \delta, \theta, \sigma, i \rangle$*, wobei*

1. $\mathcal{Q} = \{Q_1, \ldots Q_n\}$ *eine endliche Menge von nichtleeren generalisierten Knoten ist,* $Q_i = \langle \mathcal{S}_i, \theta_i, \sigma_i, i_i \rangle$ *so daß* $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ *für* $i \neq j$*,*
2. $\bar{Q}$ *der generalisierte Wurzelknoten ist,*
3. $\delta : \mathsf{Feat} \times \mathcal{Q} \rightarrow \mathcal{Q}$ *eine partielle Merkmalsfunktion ist, die einem generalisierten Knoten und einem Merkmal einen generalisierten Knoten zuweist,*
4. $\theta = \bigcup_{i=1,\ldots,n} \theta_i$ *eine totale Typfunktion ist,*
5. $\sigma = \bigcup_{i=1,\ldots,n} \sigma_i$ *die Strukturfunktion*
6. $i = \bigcup_{i=1,\ldots,n} i_i$ *eine Indexfunktion*

Die Menge der generalisierte Merkmalsstrukturen, deren generalisierte Knoten nur aus einer einelementigen Knotenmenge bestehen, und deren Indexfunktion konstant ist, ist isomorph zu $\mathcal{F}$ bezüglich Subsumption und Unifikation. Mit anderen Worten, generalisierte Merkmalsstrukturen, die lediglich eine Merkmalsstruktur enthalten, verhalten sich bezüglich Subsumption und Unifikation wie die Merkmalsstruktur, die sie enthalten.

3.3 Konstruktion von Generalisierten Merkmalsstrukturen

Generalisierte Merkmalsstrukturen werden durch sukzessives Einfügen von Merkmalsstrukturen konstruiert. Die Einfügen-Operation ist somit eine Abbildung

$$\circ : \mathcal{GF} \times \mathcal{F} \rightarrow \mathcal{GF}$$

Eine typisierte Merkmalsstruktur F kann trivialerweise in eine unterspezifizierte oder partiell unifizierte Merkmalsstruktur G umgewandelt werden. Die generalisierten Knoten in G haben eine Knotenmenge mit jeweils einem Knoten aus F. Dies wird notiert durch die Funktion $gf : \mathcal{F} \rightarrow \mathcal{G}$, welche eine Merkmalsstruktur F auf die nur F enthaltende generalisierte Merkmalsstruktur G abbildet. Das Kombinieren von n Merkmalsstrukturen in eine unterspezifizierte oder partiell unifizierte Merkmalsstruktur wird notiert als $\bigcirc_{1 \leq i \leq n} F_i$.

Ähnlich wie bei der Unifikation von Merkmalsstrukturen (siehe Definition 4) wird bei der Konstruktion der partiell unifizierten oder unterspezifizierten Merkmalsstruktur zunächst die Struktur des zugrundeliegenden Graphs konstruiert. Anschließend wird die Typinformation für jeden generalisierten Knoten angepaßt. Die Konstruktionen von partiell unifizierten und unterspezifizierten Merkmalsstrukturen unterscheiden sich lediglich in der Definition der

Merkmals-, der Typ- und der Indexfunktion. Die Konstruktion der Typ- und der Indexfunktion weist in allen drei Fällen genug Ähnlichkeiten auf, so daß ein allgemeiner Algorithmus angewandt werden kann. Dieser Algorithmus ist in Abbildung 3.4 gegeben.

3.3.1 Konstruktion des zugrundeliegenden Graphs

Der erste Schritt beim Einfügen einer Merkmalsstruktur F in eine generalisierte Merkmalsstruktur G ist die Struktur des zugrundeliegenden Graphs anzupassen. Dies geschieht in einer Weise, die sehr ähnlich ist zu dem ersten Schritt in der Unifikation von Merkmalsstrukturen (siehe Definition 4): Generalisierte Knoten Q der generalisierten Merkmalsstruktur G werden mit Knoten q der Merkmalsstruktur F in Relation gesetzt. Dazu definieren wir die Relation $\dot{\bowtie}$ zwischen generalisierten Knoten von G und Knoten von F.

Für partiell unifizierte und Merkmalsstrukturen ist die Relation $\dot{\bowtie}$ dieselbe wie die Relation $\bowtie$. Für unterspezifizierte Merkmalsstrukturen wird ein Knoten q mit einem generalisierten Knoten Q assoziiert, wenn es einen Pfad von den jeweiligen Wurzel-Element zu Q beziehungsweise q gibt. Formell ist für unterspezifizierte Merkmalsstrukturen die Relation $\dot{\bowtie}$ die kleinste Relation, für die die folgenden Bedingungen gelten:

1. $\bar{Q} \dot{\bowtie} \bar{q}$ gilt
2. wenn $Q \dot{\bowtie} q$ und es gibt a Pfad π so daß $\delta(\pi, \bar{Q}) = Q$ und $\delta(\pi, \bar{q}) = q$ und $\delta(f, Q)$ sowohl als auch $\delta(f, q)$ definiert sind, dann gilt auch $\delta(f, Q) \dot{\bowtie} \delta(f, q)$.

Die Aussage $Q \dot{\bowtie} q$ impliziert damit $Q \bowtie q$, aber der Umkehrschluß ist im allgemeinen nicht wahr. Abbildung 3.3.1 zeigt Beispiele für zwei Merkmalsstrukturen, für die $\bowtie$ und $\dot{\bowtie}$ unterschiedlich sind. Während die Beziehung $\bowtie$ zwischen den Knoten r_2, q_2 und q_3 gilt, hält die Relation $\dot{\bowtie}$ nicht zwischen q_2 and q_3, da es keinen Pfad π gibt, der von der jeweiligen Wurzel zu den Knoten führt.

Wir können dann die Bedingung zum Zusammenführen von Knotenmengen und Knoten wie folgt formulieren.

$$C_\delta(Q, q) := \begin{cases} Q \bowtie q \text{ gilt} & \text{für partiell unifizierte Strukturen,} \\ Q \dot{\bowtie} q \text{ gilt} & \text{für unterspezifizierte Strukturen} \end{cases} \tag{3.7}$$

Wie von Carpenter [1992] bemerkt, kann die Relation $\bowtie$ in Laufzeit proportional zu, $O\left(\mid V \mid + \mid E \mid ack^{-1}\left(\mid V \mid + \mid E \mid\right)\right)$ konstruiert werden, wobei V und E die Mengen der Knoten und Kanten des zugrundeliegenden Graphs bezeichnen, und ack^{-1} die inverse Ackermann Funktion ist (dieser Faktor wird von dem *Union-Find* Algorithmus beigetragen, der die Äquivalenzklassen der Knoten zusammenfügt. Die Relation $\dot{\bowtie}$ kann in Laufzeit proportional zu $O\left(\mid V + E \mid\right)$ durch paralleles Traversieren der Graphen in Tiefenordnung konstruiert werden.

Construct Generalized Feature Structure

Input: $F = \langle Q, \bar{q}, \theta, \delta \rangle$ a typed feature structure
$GF = \langle \mathcal{Q}, \bar{Q}, \theta', \delta', \sigma', i \rangle$ a generalized feature structure
Output: The modified generalized feature structure GF

1 Construction of the underlying graph and the node sets
constructgraph$(Q, Q', \delta, \delta', \mathcal{R})$
2 Construction of the generalized nodes

for each node set $R \in \mathcal{R}$
  $Q \leftarrow$ new empty generalized node
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q\}$
  for each node $r \in R$
    $i_r \leftarrow \begin{cases} i_1 & r \in Q \\ i_2 & r \in Q' \end{cases}$
    insertnode $(r, \theta(r), i_r)$
  end
end

Abbildung 3.4. Algorithmus, der eine Merkmalsstruktur in eine generalisierte Merkmalsstruktur einfügt

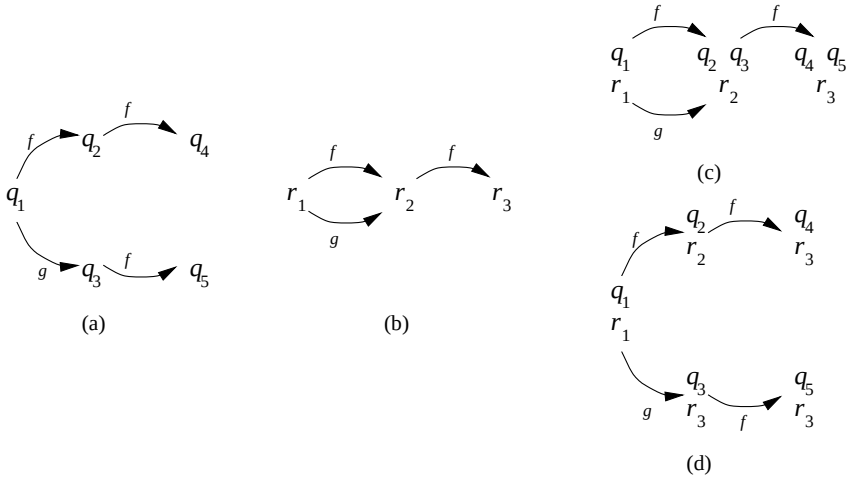


Abbildung 3.5. Zwei Merkmalsstrukturen mit unterschiedlichen Relationen $\bowtie$ und $\dot{\bowtie}$. Die beiden ursprünglichen Merkmalsstrukturen sind in (a) und (b) gezeigt. Die Relation $\bowtie$ ist in (c) gezeigt. Die Relation $\dot{\bowtie}$ ist in (d) gezeigt.

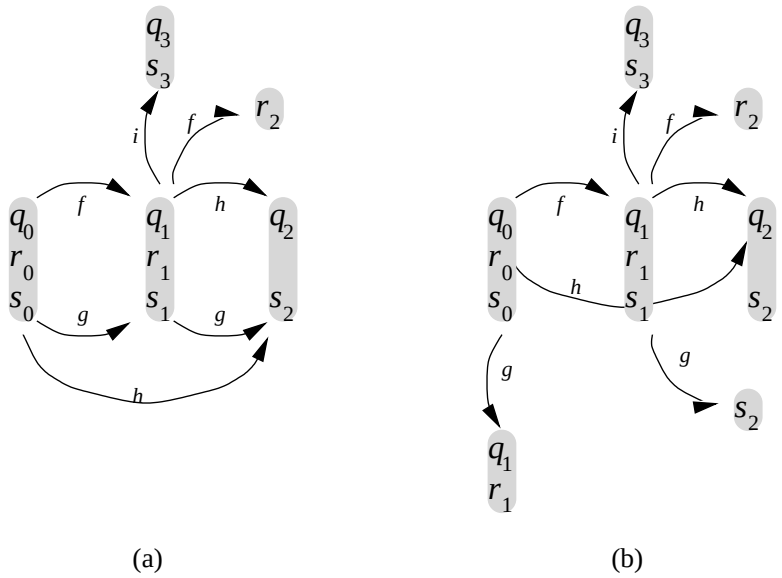


Abbildung 3.6. Die Relationen $\bowtie$ und $\dot{\bowtie}$ für die in Abbildung 2.2 gezeigten Merkmalsstrukturen.

3.3.2 Einfügen von Knoten

Nachdem der der generalisierten Merkmalsstruktur zugrundelegende Graph erweitert worden ist, werden durch die $\dot{\bowtie}$ die Knoten q von F den generalisierten Knoten Q von G zugeordnet. Es verbleibt, die Knoten q in die generalisierten Knoten Q der generalisierten Merkmalsstruktur G einzufügen, wenn immer $Q \dot{\bowtie} q$ gilt. Dies geschieht wie folgt. Wenn es in Q eine Knotenmenge S gibt, so daß $\theta(S) = \theta(q)$ gilt, dann wird q in S eingefügt. Gibt es keine solche Knotenmenge S, wird sie konstruiert, und an geeigneter Stelle in den generalisierten Knoten Q eingefügt. Schließlich muß noch die Indexfunktion i angepaßt werden. Dazu werden alle Knotenmengen S bestimmt, deren Typ $\theta(S)$ mit $\theta(q)$ kompatibel ist (im Falle von partiell unifizierten Strukturen) oder Information teilt (im Falle von unterspezifizierten Strukturen). Der Wert der Indexfunktion von diesen Knotenmengen wird um $i = index(F)$ erweitert.

Wie aus der informellen Beschreibung des Einfügens von Knoten ersichtlich, unterscheidet sich der Algorithmus für partiell unifizierte und unterspezifizierte nur in der Bestimmung der Knotenmengen S, die um den Index $i = index(F)$ erweitert werden müssen. Um diesen Unterschied explizit zu machen, formulieren wir die Bedingungen C_θ für partiell unifizierte, unterspezifizierte Strukturen als eine binäre Relation zwischen einem Knoten q und einer Knotenmenge S. C_θ ist wahr immer dann, wenn q in S enthalten sein muß.

Im Falle unterspezifizierter Merkmalsstrukturen muß q immer dann in der Knotenmenge S enthalten sein, wenn die Typinformation von q die Typinformation von S subsumiert. Mit anderen Worten, die Aussage $q \in S$ bedeutet in diesem Fall, daß der Type von q die Information des Typs von S enthält. Im Falle von partiell unifizierten Merkmalsstrukturen

Damit kommen wir zu der folgenden Bedingung:

$$C_\theta(q,S) := \begin{cases} (\theta(q) \sqsubseteq \theta(S)) & \text{für unterspezifierte Strukturen ,} \\ (\theta(q) \sqcup \theta(S) \neq \top) & \text{für partiell unifizierte Strukturen} \end{cases} \tag{3.8}$$

Der Konstruktionsalgorithmus paßt zusätzlich den Typ der Knotenmengen an, wo notwendig. Dies ist lediglich im Falle partiell unifizierter Merkmalsstrukturen erforderlich,um sicherzustellen, daß die partiell unifizierte Merkmalsstruktur zweier kompatibler Merkmalsstrukturen F, F' isomorph zu deren Unifikation $F \sqcup F'$ ist. Die Anpassungsregel für die Typen der Knotenmengen ist formuliert wie folgt:

$$type_{C_\theta}(q,q') = \sqcup\{\theta(r) : r \in Q, C_\theta(q,r) \wedge C_\theta(q',r)\}$$

Der Algorithmus zum Einfügen von Knoten ist in Abbildung 3.7 gezeigt.

Insert Node

Input: Q generalized node with root node $\bar{q} \in Q$, if $Q \neq \emptyset$
S node set $S \in Q$
q node
C condition

Output: Q modified generalized node

1 Empty Node

if $Q = \emptyset$ then
  $Q \leftarrow \{\{q\}\}$;
  return;

2 New Root Set

if $\neg C(q, root(Q))$ then
  $S' \leftarrow$ new $\emptyset$
  $\sigma(S) \leftarrow S'$
  $\theta \leftarrow \theta \cup \langle S', \theta(q) \sqcap \theta(S)\rangle$
  $i \leftarrow i \cup \langle S', i(q) \cup i(S)\rangle$
  return

3 Insert

insertNodeRec(Q,root(Q),q,C);

Abbildung 3.7. Algorithmus zum Einfügen von Knoten in einen generalisierten Knoten.

Die Idee hinter diesem Algorithmus ist, daß ein einzufügender Knoten q alle die Knotenmengen S besucht, deren Typ $\theta(S)$ kompatibel zu $\theta(q)$ (im Falle partiell unifizierter Merkmalsstrukturen) ist, oder deren Typ $\theta(S)$ $\theta(q)$ subsumiert (im Falle von unterspezifizierter Merkmalsstrukturen). Die Indexmengen aller besuchten Knotenmengen werden um den Index der Merkmalsstruktur $index(F)$ erweitert.

Im ersten Schritt des Algorithmus wird der Spezialfall eines leeren generalisierten Knotens behandelt. Im zweiten Schritt wird der Spezialfall behandelt,

Insert Node recursively

Input: Q generalized node with root node $\bar{q} \in Q$, if $Q \neq \emptyset$
S node set $S \in Q$
q node
C condition

Output: Q modified generalized node

1 Traverse Node Graph

$added \leftarrow$ false
$i(S) \leftarrow i(S) \sqcup i(q)$
for all successors S' such that $\sigma(S, S')$ holds do
  if $C(q, S')$ then
    $InsertNodeRev(Q, S', q, C)$
    $added \leftarrow$ true
if $added$ return

2 Insert Node

for all successors S' such that $\sigma(S, S')$ holds do
  if $\theta(S) \sqsubset (\theta(q) \sqcap \theta(S'))$ then
    $InsertIntermediateNode(S, S', S'', \theta(q) \sqcap \theta(S'))$
    $InsertNodeRec(S, S', S'', \theta(q) \sqcap \theta(S'))$
    $added \leftarrow$ true

if $\neg added$ then
  if $type_C\,(\theta(q), \theta(S)) = \theta(S) \vee \sigma^{-1}(S) = \emptyset$ then
    $S \leftarrow S \cup \{q\}$
    $\theta(S) \leftarrow type_C\,(\theta(q), \theta(S))$
  else
    $S' \leftarrow$ new $\{q\}$
    $\sigma^{-1}(S) \leftarrow \sigma^{-1}(S) \cup \{S'\}$
    $\theta \leftarrow \theta \cup \langle S', \theta(q) \rangle$
    $i \leftarrow i \cup \langle S', i(q) \rangle$

Abbildung 3.8. Rekursiver Teil des Einfüge-Algorithmus

$$nationality^{(1,\ldots,19)} \begin{cases} american^{(1,\ldots,9)} \\ asian^{(10,\ldots,14)} \begin{cases} chinese^{(10,\ldots,12)} \\ japanese^{(13)} \\ thai^{(14)} \end{cases} \\ european^{(15,\ldots,18)} \begin{cases} greek^{(15,16)} \\ italian^{(17,18)} \end{cases} \\ mediterranean^{(19)} \end{cases} \tag{3.9}$$

Abbildung 3.9. Ein Beispiel eines generalisierten Knotens einer unterspezifizierten Merkmalsstruktur.

wo die Relation C nicht für die Wurzelmenge S und q gilt. In diesem Fall wird eine neue Wurzelmenge eingefügt. Anderenfalls wird in die rekursive Einfügeroutine gesprungen.

Die rekursive Einfügeroutine wird nur für Knoten q und Knotenmengen S aufgerufen, für die $C_\theta(S, q)$ gilt. Die Einfügeroutine wird dann für alle Nachfolger S' in der Knotenstruktur rekursiv aufgerufen, für die die Bedingung $C_\theta(S', q)$ gilt. Damit gilt die folgende Invariante: wird beim Einfügen vom Knoten q die Knotenmenge S besucht, gilt $C_\theta(q, S)$. Deswegen muß $i(S)$ um θ erweitert werden. Gegebenenfalls müssen zwischendurch intermediäre Knoten eingefügt werden, um sicherzustellen, daß untere Schranken eindeutig bleiben. Dies geschieht durch den Aufruf von $insertIntermediateNode()$. Der Knoten q wird dann in die Knotenmenge S eingefügt, die denselben Typ trägt wie q. Existiert keine solche Knotenmenge, wird sie im letzten Schritt des Algorithmus erzeugt.

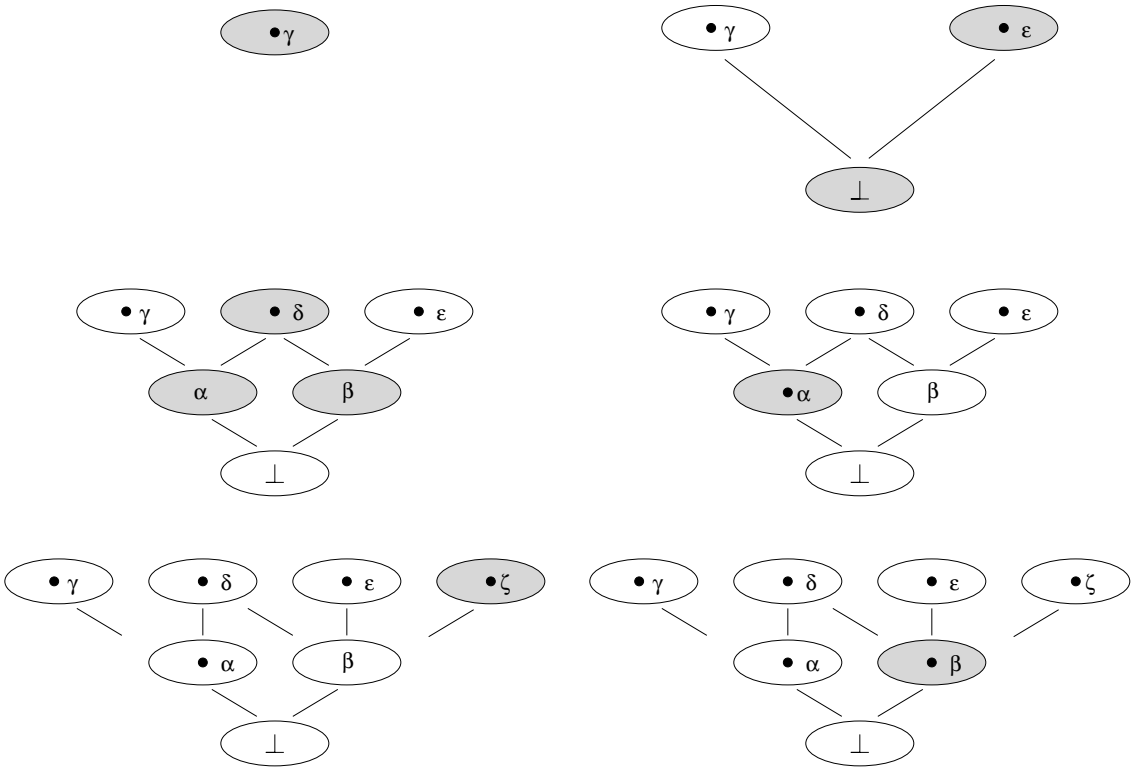


Abbildung 3.10. Konstruktion eines unterspezifizierten Knotens. Es werden Knoten mit folgenden Typen in den unterspezifizierten Knoten eingefügt: γ,ε,δ, α,ζ,β. Die Knotenmengen, die durch das Einfügen eines neuen Knotens verändert werden, sind ausgefüllt gezeichnet.

3.4 Zugriff auf Information

Die Verwendung der Generalisierten Merkmalsstrukturen im Dialog erfordert das inkrementelle Einbringen von Information in die Repräsentation. Beispiels-

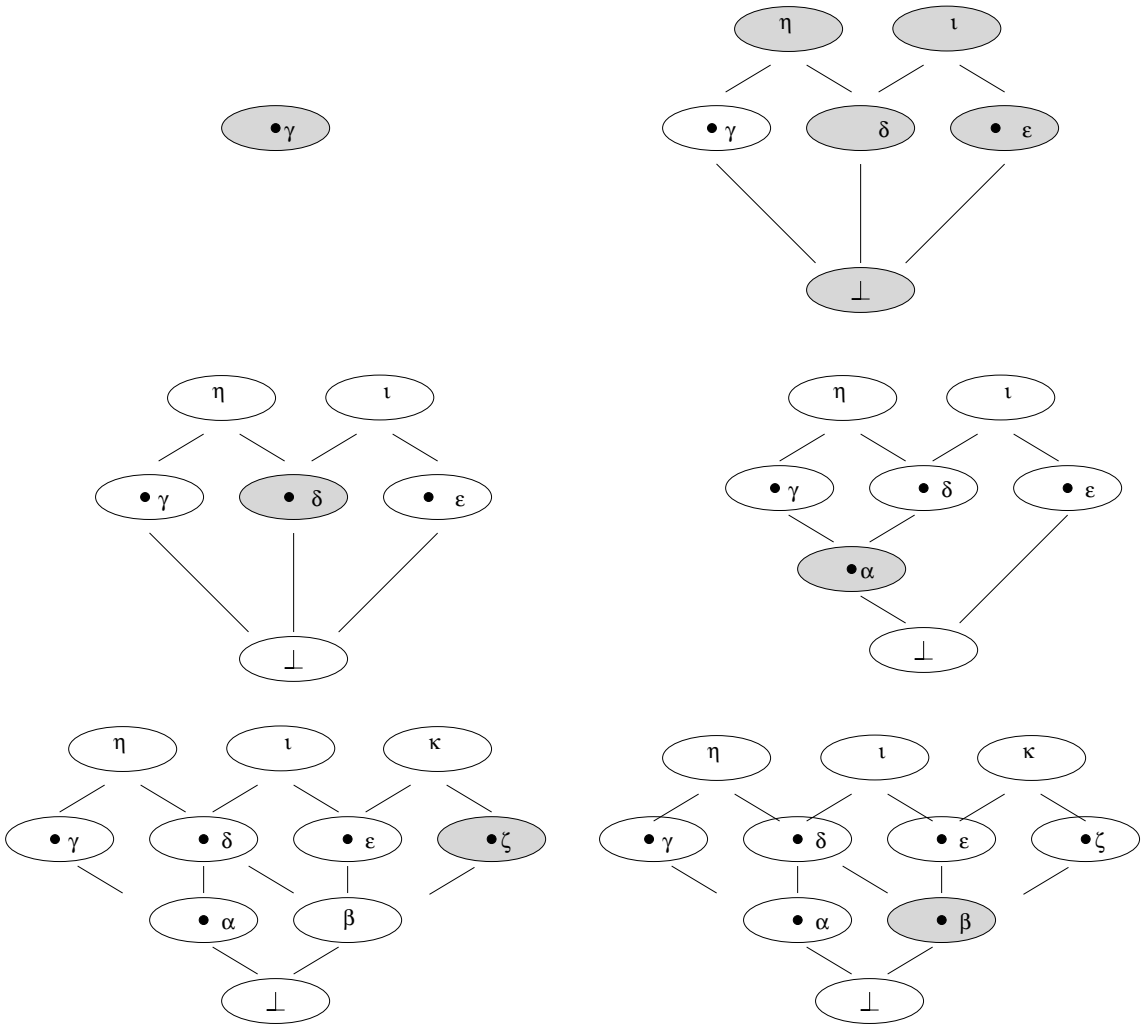


Abbildung 3.11. Konstruktion eines partiell unifizierten Knotens. Es werden Knoten mit denselben Typen wie in Abbildung 3.10 in den unterspezifizierten Knoten eingefügt: γ,ε,δ, α,ζ,β. Die Knotenmengen, die durch das Einfügen eines neuen Knotens verändert werden, sind ausgefüllt gezeichnet.

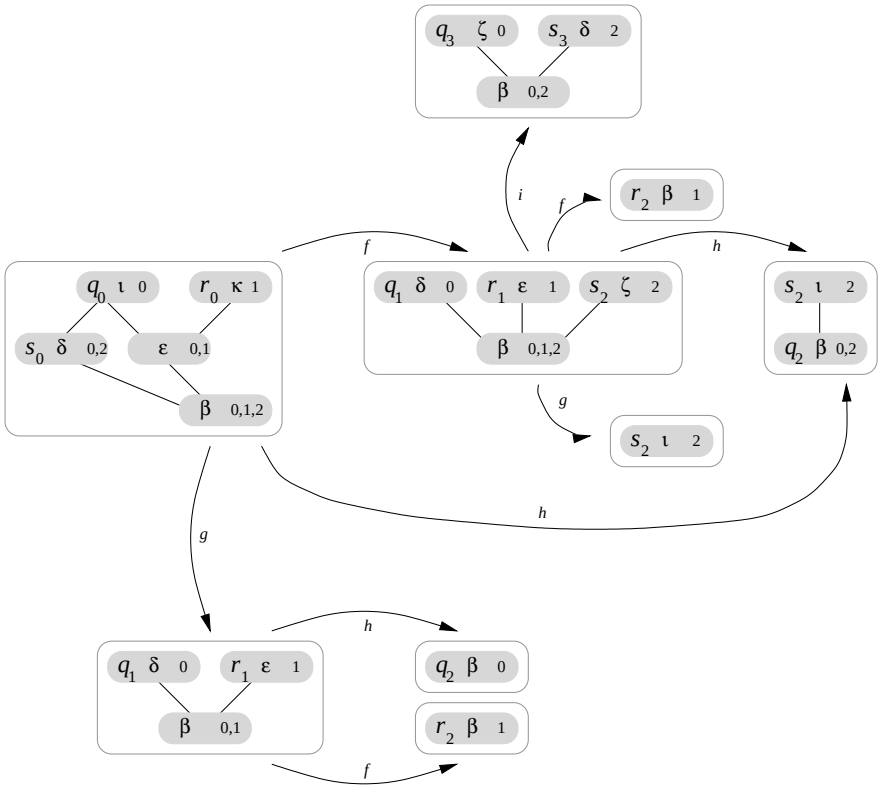


Abbildung 3.12. Die unterspezifizierte Merkmalsstruktur der drei in Abbildung 2.2 gezeigten Merkmalsstrukturen.

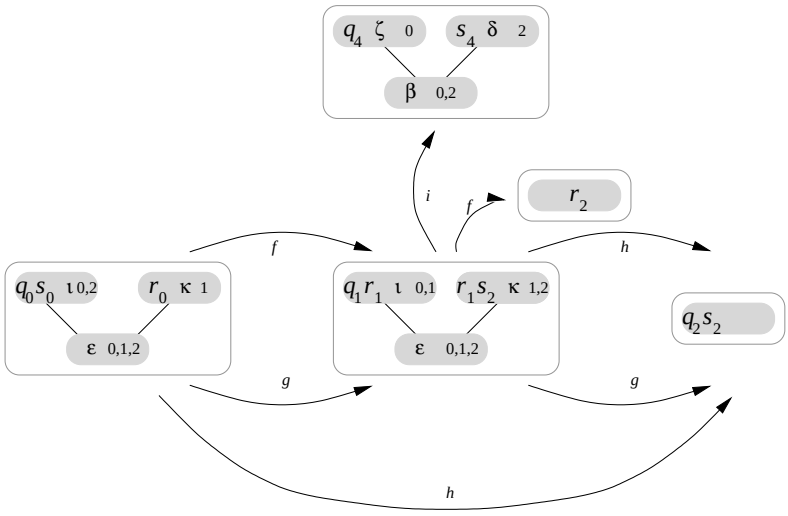


Abbildung 3.13. Die partiell unifizierte Merkmalsstruktur der drei in Abbildung 2.2 gezeigten Merkmalsstrukturen.

$$
G = \left[\begin{array}{l}
act\_givedirections^{(0-18)} \\
\left[\begin{array}{ll}
\text{ARG} & obj\_path^{(0-18)} \\
\text{DST} & \left[\begin{array}{ll}
obj\_restaurant^{(0-18)} & \\
\text{NAME} & string^{(0,\ldots,18)} \left\{\begin{array}{l}
\text{“}Gullifty's\text{“}^{(0)} \\
\text{“}Murray\ Avenue\ Grill\text{“}^{(1)} \\
\text{“}Poli's\text{“}^{(2)} \\
\text{“}Pinos'\ Pizza\ Kitchen\text{“}^{(3)} \\
\text{“}Big\ Jim's\text{“}^{(4)} \\
\text{“}Champ's\text{“}^{(5)} \\
\text{“}Pamela's\text{“}^{(6)} \\
\text{“}Silky's\text{“}^{(7)} \\
\text{“}Squirrel\ Hill\ Cafe\text{“}^{(8)} \\
\text{“}Sichuan\ House\text{“}^{(9)} \\
\text{“}Silk\ Pagoda\text{“}^{(10)} \\
\text{“}Open\ Flame\text{“}^{(11)} \\
\text{“}Susie's\text{“}^{(12)} \\
\text{“}Sweet\ Basil\text{“}^{(13)} \\
\text{“}Mediterranean\ Grill\text{“}^{(14)} \\
\text{“}Lemongrass\text{“}^{(15)} \\
\text{“}Yakov's\text{“}^{(16)} \\
\text{“}Chopsticks\ Chinese\text{“}^{(17)} \\
\text{“}Sushi\ House\text{“}^{(18)}
\end{array}\right. \\
\text{ADDRESS} & \left[\begin{array}{ll}
address^{(0,\ldots,18)} & \\
\text{CITY} & \text{“}Pittsburgh\text{“}^{(0-18)} \\
\text{NEIGHBORHOOD} & \text{“}Squirrel\ Hill\text{“}^{(0-18)} \\
\text{STREETNAME} & string^{(0,\ldots,18)} \left\{\begin{array}{l}
\text{“}Murray\ Ave\text{“}^{(0,1,2,3,7,9,11,16,17,18)} \\
\text{“}Forbes\ Ave\text{“}^{(6,8,13,14,15)} \\
\text{“}Forward\ Ave\text{“}^{(5)} \\
\text{“}Beechwood\ Boulevard\text{“}^{(10)} \\
\text{“}Shady\ Avenue\text{“}^{(12)} \\
\text{“}Saline\ Street\text{“}^{(4)}
\end{array}\right.
\end{array}\right] \\
\text{NATIONALITY} & prp\_nationality^{(0,\ldots,18)} \left\{\begin{array}{ll}
american^{(0,1,2,4,5,6,7,8,16)} & \\
asian^{(9,10,15,17,18)} & \left\{\begin{array}{l} chinese^{(9,10,17)} \\ japanese^{(18)} \\ thai^{(15)} \end{array}\right. \\
european^{(3,11-13)} & \left\{\begin{array}{l} greek^{(11,12)} \\ italian^{(3,13)} \end{array}\right. \\
mediterranean^{(14)} &
\end{array}\right. \\
\text{TYPEOFFOOD} & prp\_typeoffood^{(0,\ldots,18)} \left\{\begin{array}{l}
generic^{(0-15,17-18)} \\
vegetarian^{(16)}
\end{array}\right. \\
\text{PRICE} & prp\_price^{(0,\ldots,18)} \left\{\begin{array}{l}
moderate^{(0,1,3,9,12,13,14,17,18)} \\
cheap^{(2,4,5,6,7,8,10,11,15,16)}
\end{array}\right.
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Abbildung 3.14. Ein Beispiel einer unterspezifizierten Merkmalsstruktur.

weise entscheidet sich der Anwender, nachdem die Struktur in Abbildung 3.14 erzeugt worden ist, für asiatische Restaurants in der mittleren Preisklasse entscheidet. Nun muß eine Repräsentation der Untermenge der in Abbildung 3.14 repräsentierten Objekte bestimmt werden, die die Anforderungen des Anwenders erfüllt. Dazu werden zunächst für jeden Constraint die Indexmengen bestimmt, die den Constraint erfüllen. Im gegebenen Beispiel sind dies die Mengen $I_1 = \{9, 10, 15, 17, 18\}$ für asiatische Restaurants und $I_2 = \{0, 1, 3, 9, 12, 13, 14, 17, 18\}$ für Restaurants der mittleren Preisklasse. Die Schnittmenge $I_1 \cap I_2 = \{9, 17, 18\}$ verweist auf die Repräsentationen der Objekte, die den Anforderungen des Anwenders entsprechen. Die resultierende Repräsentation nach Entfernen aller inkompatiblen Repräsentationen ist in Abbildung 3.15 gezeigt. Dieser Vorgang des Entfernens von Knotenmengen aus G wird *Reduktion auf die Indexmenge I* genannt und wird notiert als $G\,|_I$.

$$
G' = \left[\begin{array}{l}
\mathit{act\_givedirections}^{(9,17,18)} \\
\left[\begin{array}{l}
\mathit{obj\_path}^{(9,17,18)} \\
\left[\begin{array}{ll}
\mathit{obj\_restaurant}^{(9,17,18)} & \\
\text{NAME} & \mathit{string}^{(9,17,18)} \left\{\begin{array}{l} \text{“}\mathit{Sichuan\ House}\text{“}^{(9)} \\ \text{“}\mathit{Chopsticks\ Chinese}\text{“}^{(17)} \\ \text{“}\mathit{Sushi\ House}\text{“}^{(18)} \end{array}\right. \\
\text{ADDRESS} & \left[\begin{array}{ll} \mathit{address}^{(9,17,18)} & \\ \text{CITY} & \text{“}\mathit{Pittsburgh}\text{“}^{(9,17,18)} \\ \text{NEIGHBORHOOD} & \text{“}\mathit{Squirrel\ Hill}\text{“}^{(9,17,18)} \\ \text{STREETNAME} & \mathit{Murray\ Ave}\text{“}^{(9,17,18)} \end{array}\right] \\
\text{NATIONALITY} & \mathit{asian}^{(9,17,18)} \left\{\begin{array}{l} \mathit{chinese}^{(9,17)} \\ \mathit{japanese}^{(18)} \end{array}\right. \\
\text{TYPEOFFOOD} & \mathit{generic}^{(9,17,18)} \\
\text{PRICE} & \mathit{moderate}^{(9,17,18)}
\end{array}\right]
\end{array}\right] \text{DST}
\end{array}\right] \text{ARG}
$$

Abbildung 3.15. Die unterspezifizierte Merkmalsstruktur, eingeschränkt auf Restaurants der nationalität *prp_asian* und der Preisklasse *moderate*.

Um das Verfahren des inkrementellen Einbringens von Information formell zu beschreiben, betrachten wir *Typ-Constraints* der Form $\pi : \tau$, wobei π ein Merkmalspfad und τ ein Typ ist. Die *Lösungsmenge* $L(G, \pi : \tau)$ eines Typconstraints bezüglich einer unterspezifizierten (partiell unifizierten) Merkmalsstruktur G ist definiert als die Menge von Indices im generalisierten Knoten an der Stelle π, deren Typen von τ subsumiert werden (zu τ kompatibel sind). Im Abbildung 3.15 gilt beispielsweise folgende Gleichung:

$$L(G', \text{ARG}|\ \text{DST}\ |\ \text{NATIONALITY} : \mathit{chinese}) = \{9, 17\} \tag{3.10}$$

da die Knotenmenge mit dem Typ *chinese* die Indexmenge $\{9, 17\}$ trägt.

Die Lösungmenge $L(G, \{\pi_1 : \tau_1, \ldots, \pi_n : \tau_n\})$ einer Menge von Typ-Constraints ist die Schnittmenge $L(G, \pi_1 : \tau_1) \cap \ldots \cap \ldots, L(G, \pi_n : \tau_n)$. Die

Lösungsmenge $L(G, \{\pi_1 : \tau_1, \ldots, \pi_n : \tau_n\})$ enthält dann genau die Indices der Merkmalsstrukturen aus G, die die Information $\pi_i : \tau_i$ enthalten (mit der Information $\pi_i : \tau_i$ kompatibel sind).

Wie im obigen Beispiel gezeigt, kann eine generalisierte Merkmalsstruktur mittels einer Indexmenge reduziert werden. Die Anwendung einer Lösungsmenge $L(G, \{\pi_1 : \tau_1, \ldots, \pi_n : \tau_n\})$ auf eine generalisierte Merkmalsstruktur G resultiert in einer generalisierten Merkmalsstruktur G', die dann nur die Merkmalsstrukturen enthält, deren Indices in der Lösungsmenge sind.

Darüber hinaus können partiell unifizierte Merkmalsstrukturen auch dazu verwendet werden, Informationen nichtmonoton aus einer Merkmalsstruktur zu entfernen. Hierzu kann ausgenutzt werden, daß partiell unifizierte Merkmalsstrukturen inkompatible Informationen separieren. Sei F_i eine Merkmalsstruktur im Diskurs, und F_j die Repräsentation einer Information, mit der F_i überschrieben werden soll. Die partiell unifizierte Merkmalsstruktur $G = gf(F_i) \circ F_j$ separiert dabei die Informationen in F_i und F_j, die zueinander kompatibel sind, von miteinander inkompatibler Information. Anschließend kann dann die resultierende partiell unifizierte Merkmalsstruktur auf die Indexmenge $\{j\}$ eingeschränkt werden.

Ein Beispiel soll dieses Vorgehen erläutern. Seien F_i und F_j gegeben durch die Merkmalsstrukturen

$$F_i = \begin{bmatrix} \textit{obj\_restaurant} \\ \text{NATIONALITY} & \textit{prp\_chinese} \\ \text{PRICE} & \textit{prp\_expensive} \end{bmatrix}$$

$$F_j = \begin{bmatrix} \textit{obj\_restaurant} \\ \text{NATIONALITY} & \textit{prp\_japanese} \end{bmatrix}$$

Die resultierende partiell unifizierte Merkmalsstruktur ist gegeben durch[3]

$$\begin{aligned} G = \; & gf(F_i) \circ F_j \\ = \; & \begin{bmatrix} \textit{obj\_restaurant}^{(i,j)} \\ \\ \text{NATIONALITY} & \textit{asian}^{(i,j)} \begin{cases} \textit{chinese}^{(i)} \\ \textit{japanese}^{(j)} \end{cases} \\ \\ \text{PRICE} & \textit{prp\_expensive}^{(i,j)} \end{bmatrix} \end{aligned}$$

[3] In diesem Falle unterscheidet sich die partiell unifizierte Merkmalsstruktur von F_i und F_j von der unterspezifierten Merkmalsstruktur von F_i und F_j dadurch, daß die Indexmenge des Typs *prp_expensive* nur den Index i enthält, und nicht die beiden Indices i und j. Dies ist begründet durch die Tatsache, daß unterspezifizierte Merkmalsstrukturen *gemeinsame* Information isolieren, während partiell unifizierte Merkmalsstrukturen *kompatible* Information isolieren. Dies demonstriert desweiteren die Notwendigkeit beider Operationen.

und nach der Einschränkung auf die Indexmenge $\{j\}$ wird die folgende Merkmalsstruktur erzeugt

$$
\begin{aligned}
F = &\ G\mid_{\{j\}} \\
= &\ (gf(F_i) \circ F_j)\mid_{\{j\}} \\
= &\begin{bmatrix} \textit{obj\_restaurant} & \\ \text{NATIONALITY} & \textit{prp\_japanese} \\ \text{PRICE} & \textit{prp\_expensive} \end{bmatrix}
\end{aligned}
$$

Diese Operation kann also verwendet werden, um interaktive Korrekturen auf den Repräsentationen auszuführen. Wir schreiben $F_i - F_j$ als Abkürzung für $(gf(F_i) \circ F_j)\mid_{\{j\}}$.

3.5 Subsumption, Unifikation und Wohltypisiertheit

Generalisierte Merkmalsstrukturen sind, wie oben beschrieben, Datenstrukturen, die Merkmalsstrukturen enthalten. Damit ist es einfach, die von den typisierten Merkmalsstrukturen bekannten Konzepte von Subsumption, Unifikation, Wohltypisiertheit und Typeninferenz auf generalisierte Merkmalsstrukturen zu übertragen.

3.5.1 Subsumption

Eine generalisierte Merkmalsstruktur $G = \bigcirc_i^n F_i$ subsumiert eine generalisierte Merkmalsstruktur $G' = \bigcirc_j^m F'_j$ genau dann wenn es eine Merkmalsstruktur F_i gibt, so daß $F_i \sqsubseteq F'_j$ gilt für alle $1 \leq j \leq m$.

3.5.2 Unifikation

Die Unifikation zweier unterspezifizierter (partiell unifizierter) Merkmalsstrukturen $G = \bigcirc_i^n F_i$ und $G' = \bigcirc_i^m F'_i$ ist gegeben durch die Unterspezifikation (partielle Unifikation) der paarweisen Unifikationen der in G und G' enthaltenen Merkmalsstrukturen, oder formell:

$$G \sqcup G' = \bigcirc_{i,j}\{F_i \sqcup F'_j : F_i \sqcup F'_j \text{ ist definiert } \} \tag{3.11}$$

Die Unifikation $G \sqcup G'$ ist undefiniert, wenn $F_i \sqcup F'_j$ überall undefiniert ist.

3.5.3 Typinferenz

Typinferenz kann analog auf generalisierte Merkmalsstrukturen erweitert werden. Eine generalisierte Merkmalsstruktur G heißt wohltypisiert, wenn alle in G enthaltenen Merkmalsstrukturen wohltypisiert sind. Das Ergebnis einer auf G angewandten Typinferenzprozedur ist definiert als die Unterspezifikation (partielle Unifikation) der Typinferenzprozedur, angewandt auf die in G enthaltenen Merkmalsstrukturen.

3.6 Diskussion

In der Vergangenheit wurden mehrere Ansätze vorgeschlagen, mittels derer Merkmalsstrukturen verallgemeinert werden können. Allerdings war die Motivation für diese Ansätze eine andere als im vorliegenden Fall. Ein Problem, daß bei der Spezifikation lexikalischer Einträge auftritt, ist die Tatsache, daß die Abbildung von der Oberflächenform eines Wortes auf den lexikalischen Eintrag oft nicht eindeutig ist. Um die darus entstehende Komplexität bei der Satzanalyse (und den erhöhten Aufwand bei der Spezifikation der Einträge) zu reduzieren, wurden verschiedene Varianten von *disjunktiven Merkmalsstrukturen* vorgeschlagen [Dörre and Eisele, 1990,Eisele and Dörre, 1992,Miyao, 1999]. Die vorgeschlagenen Lösungen führen alle eine Notation für Disjunktion und ermöglichen es, mehrere Disjunkte zu "synchronisieren". *Named disjunctions* beispielsweise synchronisieren die Seiten des Oder-Symbols. In

$$\begin{bmatrix} a & \\ \text{F} & a \\ \text{G} & b \end{bmatrix} \quad \vee \quad \begin{bmatrix} a & \\ \text{F} & c \\ \text{G} & d \end{bmatrix} \quad = \begin{bmatrix} a & \\ \text{F} & a \vee_1 c \\ \text{G} & b \vee_1 d \end{bmatrix} \tag{3.12}$$

wird somit die Kombination

$$\begin{bmatrix} a & \\ \text{F} & a \\ \text{G} & d \end{bmatrix}$$

ausgeschlossen, die nicht von den Merkmalsstrukturen auf der linken Seite der Gleichung unterstützt wird. Es wurden mehrere Vorschläge [Eisele and Dörre, 1992,Miyao, 1999]) gemacht, wie Unifikation auf disjunktive Merkmalsstrukturen zu erweitern ist. In den hier vorgeschlagenen unterspezifizierten Merkmalsstrukturen erlauben die Indexfunktionen das Auffinden zusammengehörender Information.

In anderen Arbeiten wurde untersucht, wie Default-Information in Merkmalsstrukturen repräsentiert werden kann [Bouma, 1990], [Rounds and Zhang, 1991], [Bouma, 1992], [Carpenter, 1993] [Lascarides and Copestake, 1996], [Lascarides and Copestake, 1999]. Wenn Default-Informationen auch prinzipiell von disjunktiver Information verschieden ist, müssen doch ähnliche Probleme gelöst werden. Um Default-Information zu einem späteren Zeitpunkt im Schlußprozeß zurücknehmen zu können, darf eine Default-Unifikationsprozedur niemals eine Repräsentation erzeugen, die die Form der Argument "vergißt". Aus diesem Grunde erweitern Lascarides und Copestake [1999] typisierte Merkmalsstrukturen um einen *tail*, dessen Aufgabe es ist, die Argumente zu Unifikationsoperationen abzuspeichern. Zu jedem gegebenen Zeitpunkt können dann die Elemente mit maximaler Priorität des *tails* in die Merkmalsstruktur inkorporiert werden, die nicht widersprüchlich sind. Das hier beschriebene Vorgehen im Falle der partiell unifizierten Merkmalsstrukturen ist ähnlich, da jeder Knoten der in der partiellen Unifikation partizipierenden Merkmalsstrukturen seinen Typ während des gesamten Prozesses behält.

3.7 Zusammenfassung

In diesem Kapitel wurden generalisierte Merkmalsstrukturen vorgestellt. Generalisierte Merkmalsstrukturen ermöglichen es, eine Menge von n typisierten Merkmalsstrukturen zu repräsentieren. Dabei gibt es zwei verschiedene Repräsentationsformen. Unterspezifizierte Merkmalsstrukturen repräsentieren die Gemeinsamkeiten der n typisierten Merkmalsstrukturen, während partiell unifizierte Merkmalsstrukturen die Information darstellen, die kompatibel mit den n typisierten Merkmalsstrukturen ist.

Die Anwendungen für Dialogverarbeitung sind die Auswahl des semantischen Inhaltes von Klärungsfragen und interaktives Verbessern.

Kapitel 4

Sprachverarbeitungsdienste

Natürlichsprachliche Dialogverarbeitungssysteme müssen in scheinbar intelligenter Weise auf sprachliche Äußerungen des Benutzer reagieren. Damit ein sinnvoller, zweckgerichteter Austausch von Information stattfindet, müssen wenigstens die folgenden Schritte auf Seiten des Dialogsystems ausgeführt werden:

(i) **Analyse**
die natürlichsprachliche Eingabe, wie sie vom Spracherkenner transkribiert wird, muß analysiert und in eine geeignete semantische Repräsentation umgeformt werden,

(ii) **Entscheidungsfindung**
basierend auf der semantischen Repräsentation, dem Kontext und der Anwendung, eine Entscheidung, ob und welche Information dem Benutzer übermittelt werden soll, sowie ob und welche Aktion auf Seiten der Anwendung ausgeführt werden soll, muß getroffen werden,

(iii) **Generation**
die zu übermittelnde Information wird in natürliche Sprache transformiert und dem Benutzer mitgeteilt.

Diese Schritte, Analyse, Entscheidungsfindung unter Benutzung von Hintergrundwissen, und Generation, sind ähnlich denen, die in jedem wissensbasierten System ausgeführt werden. Im Falle von natürlichsprachliche Dialogverarbeitungssystemen verlangen die Schritte (i) und (iii) die Verwendung von Sprachverarbeitungsalgorithmen wie Parsing und Generierung natürlicher Sprache.

Insbesondere der erste Schritt der Analyse erfordert fehlertolerante Algorithmen, da weder eine korrekte Transkribierung der Äußerung durch den Spracherkenner noch grammatikalische Sätze von Seiten des Benutzers garantiert werden können. In ersten Systemen wurde deswegen eine *Key Word Spotting* genannte Technik verwandt, um die Bedeutung der gesprochenen Sätze zu analysieren. Hier wurde die Äußerung auf wenige Schlüsselwörter untersucht, ohne die gesamte Satzstruktur aufzubauen. In dieser Arbeit wird der semantische Parser Soup [Gavalda, 2000] verwendet, eine Weiterentwicklung des Parsers Phoenix [Ward, 1994]. Der Parser verwendet robuste semantische Grammatiken, die, je nach Anforderung der Anwendung, einen fließenden Übergang zwischen Schlüsselwortparsing und vollständig kontextfreien Ableitungen erlauben.

Generierung der Sprachausgabe in frühen Dialogsystemen wurde zumeist durch Schablonen realisiert. Allen [1996] bemerkt, daß keine komplexe Generierungskomponente nötig sei, da die Rückmeldungen meist kurz sind und Variabilität bei der Satzkonstruktion nicht benötigt wird. Walker et al [2001] bemerkt, daß mit zunehmender Komplexität der Anwendung schablonenartige Generierung aufwendig zu erstellen ist. Sie schlagen einen kompletten Sprachgenerierungsprozeß, bestehend aus Textplaner, Satzplaner und Satzrealisierer vor. Der Satzplaner generiert dabei eine Menge von 20 Satzplänen, von denen der optimale durch eine mit einem *Boost* Algorithmus gelernte Optimierungsfunktion ausgewählt wird. Der ausgewählte Satzplan wird dann mithilfe einer lexikalischen Grammatik realisiert. Der hier vorgeschlagene komplexere Ansatz ist motiviert durch die elaborierte Rückmeldung (engl. *implicit confirmation*) der vom Anwender bereitgestellten Information.

4.1 Einleitung

In diesem Kapitel wird die Implementierung der Schritte (i) Satzanalyse und (iii) Generierung beschrieben. Insbesondere wird beschrieben, wie allgemeine Parsing- und Generierungsdienste durch die Abstrakte Dialogebene zur Verfügung gestellt werden können.

Die Dienste zur Unterstützung der Satzanalyse konvertieren die Ausgabe des Spracherkenners in eine typisierte Merkmalsstruktur. Dabei werden kontextfreie Grammatikregeln verwendet, die mit Fragmenten von typisierten Merkmalsstrukturen annotiert sind. Die resultierende Merkmalsstruktur setzt sich dann kompositionell aus den Fragmenten der im Ableitungsbaum verwendeten Regeln zusammen. Da die Fragmente der Merkmalsstrukturen ebenso wie die Nichtterminalsymbole der Grammatik typisiert sind, kann eine Typüberprüfung bereits zur Übersetzungszeit des Systems stattfinden.

Um Klärungsfragen zu generieren, wurde eine einfache schablonenbasierte Generierungsmethode implementiert. In ersten Ansätzen bei der Implementierung des vorliegenden Systems wurde versucht, grammatikbasierte Gemerierung unter Wiederverwendung der Parsinggrammatiken zu implementieren. Es ist jedoch schwierig, reversible Grammatiken so zu schreiben, daß sie sowohl die nötige Robustheit beim Parsen und gleichzeitig die entsprechende Genauigkeit beim Generieren aufweisen müssen. Deswegen wurde der einfachere, aber besser zu kontrollierende Ansatz der schablonenbasierten Generierung verfolgt. Um den Argumenten von Walker et al [2001] zu begegnen, wurden die Schablonen nach Dialogzustand (siehe Kapitel 7) klassifiziert. Damit ist zwar eine so flexible Generierung, wie sie durch ein eigenständiges Generierungssystem gegeben ist, nicht möglich. Auf der anderen Seite ist die Generierung flexibel genug, um vom Anwender bereitgestellte Information flexibel zu bestätigen. Desweiteren wird keine eigenständige Grammatik für die Generierung benötigt.

Es ist Ziel der Arbeit, die Entwicklung neuer natürlichsprachliche Anwendungen durch Kombination von Wissensquellen bestehender Anwendungen zu beschleunigen. Um Kollisionen von Definitionen zu vermeiden, können alle verwendeten Wissensquellen in eigenen Namensräumen angegeben werden. Nach

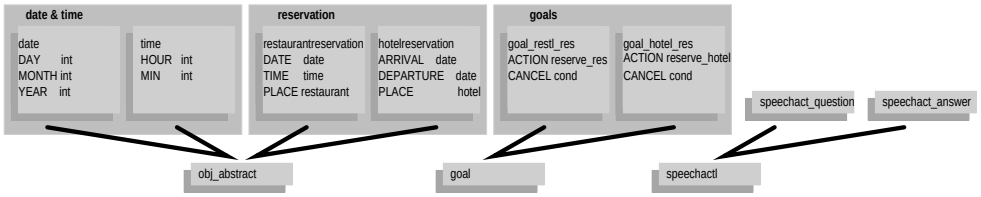


Abbildung 4.1. Ein Teil einer Typenhierarchie mit Wohlgeformtheitsspezifikationen. Zwei generische Untermodelle, date & time und reservations genannt, werden mit anwendungsspezifischen Dialogzielspezifikationen verbunden. Der Teil der Typenhierarchie, der die Sprechakte enthält, ist domänenunabhängig.

der Einführung der Namensräume werden in diesem Kapitel dann die Satzanalyse und die natürlichsprachliche Generierung vorgestellt.

4.2 Namensräume

Da die Typenhierarchien für eine gegebene Anwendung aus mehreren Unterdomänen bestehen kann, von denen jede möglicherweise eine wiederverwertbare Komponente darstellt, werden die Typen und Merkmale in eigenen Namensräumen deklariert. Damit werden Namenskonflikte zwischen verschiedenen Komponenten vermieden, die gleichzeitig verwendet werden sollen. Ebenso sind die anderen Wissensquellen, wie Dialogzielbeschreibungen, Grammatikregeln und Generierungsschablonen in Namensräumen deklariert.

4.3 Grammatiken

Gegenwärtige natürlichsprachliche Anwendungen sind typischerweise auf eine semantisch wohldefinierte Domäne beschränkt. Diese Einschränkung erlaubt es, Annahmen über die Form der nötigen Ableitungen zu machen, die die Grammatik vereinfachen. Gleichzeitig weisen Grammatiken in einer Sprache, die für eine Domäne geschrieben wurden, große strukturelle Ähnlichkeiten mit einer Grammatik für dieselbe Sprache, aber eine andere Domäne auf. Zum Beispiel decken viele Dialoggrammatiken Satzfragmente zum Anfordern von Hilfe, Einleitungen zu Aufforderungen, wie zum Beispiel *"Ich möchte . . . ", "Ich würde gerne . . . "* ab. Da diese Grammatikfragmente aufwendig zu entwickeln sind, ist es wünschenswert, sie wiederverwenden zu können. Allerdings erfordert Wiederverwendbarkeit von diesen Grammatikregeln Modularität unterhalb der Grammatikregelebene, da nur Teile von Grammatikregeln wiederverwendet werden können.

Es hat zwar in der Vergangenheit Studien zu Wiederverwendbarkeit oder Modularisierung von Grammatiken gegeben [Levin *et al.*, 1998]. Allerdings wurden bis jetzt keine objekt-orientierten Methoden entwickelt, die Techniken wie Vererbung und Namensräume zur Kombination von Grammatiken und Ontologien verwendet. In diesem Kapitel werden von daher Prinzipien zur Vererbung von Grammatikregelfragmenten entwickelt, die insbesondere in Kombination mit der IS-A Relation der semantischen Konzepte in der Ontologie eine höhere Wiederverwendung von Sub-Grammatiken erlaubt.

4.3.1 Vektorisierte Kontextfreie Grammatiken

Grammatiken für aufgabenorientierte Dialogsysteme bestehen typischerweise aus semantischen Grammatiken. Die Nichtterminalsymbole semantischer Grammatiken kodieren typischerweise syntaktische und semantische Information. Die Regel

req_directions → `wie komme ich zu`
| `zeig mir den weg zu;`

zum Beispiel interpretiert die rechtsstehenden Satzfragmente als eine Aufforderung zur Anzeige des Weges auf einer Karte. Die Nichtterminalsymbole dienen dabei als Interpretation.

Eine klare syntaktische Beschreibung des Satzfragments ist in der Regel im Nichtterminalsymbol nicht enthalten. Es ist von daher schwer, Grammatikfragmente in anderen Anwendungen wiederzuverwenden.

In verschiedenen Ansätzen zu robustem Parsing mit lexikalisierten Grammatiken [Vogel and Cooper, 1995] wird die Unifikation von Merkmalsstrukturen so modifiziert, daß verletzte Constraints nicht notwendigerweise zum Abbruch des Parsevorgangs führen. Lexikalisierte Grammatiken wie zum Beispiel HPSG erlauben das Trennen von syntaktischem und semantischem Wissen und würden von daher die Wiederverwendung von Substrukturen erlauben. Allerdings ist das Parsing mit unifikationsbasierten Grammatiken zeitaufwendig und die Entwicklung der Grammatiken erfordert linguistisches Wissen.

Aus diesen Gründen wurde ein Grammatikformalismus, im folgenden *Vektorisierte Kontextfreie Grammatiken* genannt, entwickelt, der einen Kompromiß zwischen semantischen Grammatiken und lexikalisierten Grammatiken darstellt. Vektorisierte kontextfreie Grammatiken sind eine einfache Generalisierung von kontextfreien Grammatiken, wobei die Nichtterminalsymbole aus n-dimensionalen Vektoren über partiell geordneten Elementen bestehen. Dies hat die folgenden Vorteile. Erstens erfolgt eine Trennung des durch das Nichtterminalsymbol ausgedrückte semantischen und syntaktischen Wissen. Zum anderen, da die Elemente in den Vektoren, die dieses Wissen repräsentieren, partiell geordnet sind, können Aspekte der Reprasentation unterspezifiziert werden. Dies ermöglicht es, Grammatikregeln mithilfe von Vererbungstechniken zu rekombinieren. Im folgenden wird genauer auf die Komponenten vektorisierter Grammatiken eingegangen.

Vektorisierte Nichtterminalsymbole. Wir definieren ein vektorisiertes Nichtterminalsymbol wie folgt.

Definition 1 (Vektorisiertes Nichtterminalsymbol). *Ein vektorisiertes Nichtterminalsymbol besteht aus einem n-dimensionalen Vektor $\langle \tau^1, \ldots, \tau^n \rangle$, wobei $\tau_i \in V_i$ und $\langle V_i, \leq \rangle$ eine partiell geordnete Menge ist.*

Der Tradition semantischer und lexikalisierter Grammatiken folgend, repräsentieren die Elemente der Nichtterminalsymbole syntakische und semantische Information. Elemente, die semantische Information repräsentieren, werden dabei durch einen Unterstrich gekennzeichnet. Semantische Information kann

beispielsweise durch Typen der Typenhierarchie repräsentiert werden. Die Symbole

$\langle \underline{obj\_trainstation}, N, sg \rangle$
$\langle \underline{act\_go}, V, non3rdsg \rangle$

beispielsweise repräsentieren gültige Nichtterminalsymbole.

Die Kombinierung von verschiedenen Nichtterminalsymbolen ist die formelle Basis für Vererbung von Grammatikregeln. Im folgenden werden Subsumption und Unifikation auf Nichtterminalsymbolen definiert. Damit die Unifikation eindeutig ist, wird gefordert, daß, wenn für zwei beliebige Elemente einer partiellen Ordnung V_i eine obere Schranke existiert, diese eindeutig ist. Dies ist in Analogie zu den Anforderungen an die Typenhierarchie in Abschnitt 2.2. Subsumption und Unifikation ist dann die Erweiterung der Subsumption und Unifikation der partiell geordneten Mengen auf die Elementvektoren.

Definition 2 (Subsumption und Unifikation). *Ein vektorisiertes Nichtterminalsymbol* $\mathbf{nt}_1 = \langle \tau^1, \ldots, \tau^n \rangle$ subsumiert *ein anderes* $\mathbf{nt}_2 = \langle \sigma^1, \ldots, \sigma^n \rangle$, *oder* $\mathbf{nt}_{\sqsubseteq}\mathbf{nt}_2$ *genau dann, wenn* $\tau_i \sqsubseteq_i \sigma_i$. *Die Unifikation von* $\mathbf{nt}_1$ *und* $\mathbf{nt}_2$, $\mathbf{nt}_1 \sqcup \mathbf{nt}_2$, *ist gegeben durch* $\langle \tau^1 \sqcup_1 \sigma^1, \ldots, \tau^n \sqcup_n \sigma^n \rangle$, *wenn* $\tau^i \sqcup_i \sigma^i$ *existiert für alle* i, *und undefiniert anderenfalls.*

Zwei Nichtterminalsymbole heißen inkompatibel, wenn deren Unifikation nicht existiert. Die Unifikationen der Nichtterminalsymbole

$\langle obj\_trainstation, N, sg \rangle \sqcup \langle obj\_trainstation, N, pl \rangle$ (a)
$\langle obj\_trainstation, N, sg \rangle \sqcup \langle act\_givedirections, V, \_ \rangle$ (b)

beispielsweise sind undefiniert.

Es ist das Ziel einer Interaktion mit einem natürlichsprachlichen Dialogsystem, einen Dienst des Systems aufzurufen. Um die Abbildung von Interpretation der Eingabesätze auf den intendierten Dienst zu erleichtern, wird gefordert, daß zwei natürlichsprachliche Eingaben, wenn sie *bezüglich der angebotenen Dienste* dieselbe Bedeutung haben, dieselbe Reprasentation zugewiesen bekommen. Um diese Anforderung zu erfüllen, muß die Interpretation der Nichtterminalsymbole manchmal etwas weitläufig ausgelegt werden. Ein Beispiel wird dies verdeutlichen. Die obigen Beispiele der Nichtterminalsymbole suggerieren, daß zwischen den syntaktischen und semantischen Kategorien eine Korrespondenz besteht, zum Beispiel eine Aktion wird durch eine Verbalphrase ausgedrückt, ein Objekt durch eine Nominalphrase und so weiter. Datensammlungen in der Domäne zeigen aber, daß dies häufig nicht der Fall ist. Die Äußerungen

Wo kann ich hier denn etwas essen gehen
Wo finde ich eine Gaststätte hier in der Nähe

haben bezüglich der Dienste eines Stadtinformationskiosks dieselbe Bedeutung, nämlich Restaurants in der Nähe des Standortes anzuzeigen. Die für diesen Zweck gewählte Repräsentation ist

$$\begin{bmatrix} act\_displayobj \\ \text{ARG } obj\_restaurant \end{bmatrix}$$

Offensichtlich ist der semantische Beitrag der Verbalphrase *etwas essen gehen* derselbe wie der der Nominalphrase *eine Gaststätte hier in der Nähe*. Demzufolge wird ein semantisches Objekt einmal durch eine Verbalphrase und einmal durch eine Nominalphrase erzeugt. Es ist demzufolge also notwendig, Grammatiken pragmatischßu schreiben.

Al Alternative dazu (die in dieser Arbeit nicht verfolgt worden ist, aber siehe zum Beispiel [Woszczyna *et al.*, 2000]) kann auch eine exakte semantische Repräsentation der Eingabe erstellt werden, um dann darauf aufbauend in einem zweiten Schritt eine aufgabenabhängige semantische Repräsentation zu erstellen (im Bereich der Maschinellen Übersetzung wird so die Interlingua-Reprasentation erzeugt). Dieser Ansatz wurde hier nicht weiter verfolgt, da das Erstellen der Abbildungen zwischen den Repräsentation nicht besonders gut für Rapid Prototyping geeignet ist.

Vektorisierte Grammatikregeln. Die vektorisierten Phrasenstrukturregeln bestehen, wie kontextfreie Phrasenstrukturregeln auch, aus Alternativen von Sequenzen von Terminal- oder Nichtterminalsymbolen. Je nach Form der Regel wird unterschieden zwischen lexikalischer und struktureller Grammatikregel. Um das Regelschreiben zu vereinfachen, können Grammatikregeln mit "*" (bevorstehendes Symbol wird keinmal oder einmal instantiiert) oder "+" (bevorstehendes Symbol wird mindestens einmal instantiiert) annotiert werden

Im folgenden wird angenommen, daß nur das Element τ^1 eines Nichtterminalsymbols $\mathbf{nt} = \langle \underline{\tau}^1, \ldots, \tau^n \rangle$ semantische Information enthält. Ist **nt** das Symbol der linken Seite, nennen wir τ^1 den *semantischen Typ* der Regel.

Lexikalische Regeln

Die rechte Seite der lexikalischen Regeln besteht aus einer nichtleeren Menge von Alternativen von Sequenzen von Terminalsymbolen.

$$\begin{array}{rl} \langle \underline{\tau}^1, \ldots, \tau^n \rangle \rightarrow & w_{11} \cdots w_{1n_1} \mid \\ \vdots & \vdots \mid \\ & w_{m1} \cdots w_{1n_m} \end{array}$$

Strukturelle Regeln

Die rechte Seite von strukturellen Regeln besteht aus einer nichtleeren Menge von Alternativen von Sequenzen von Nichtterminalsymbolen.

Semantische Konvertierungsinformation

Die Grammatikregeln sind mit semantischer Information annotiert, sodaß eine semantische Repräsentation automatisch aus dem Parsebaum erzeugt werden kann. Jedes Symbol auf der rechten Seite einer Grammatikregel kann dabei mit einem Merkmalspfad und einem Typ $\pi : \tau$ annotiert werden.

Eine strukturelle Regel sieht demnach aus wie folgt:

$$\langle \underline{\tau}^1, \ldots \tau^n \rangle \rightarrow \left[\begin{array}{c} \langle \underline{\tau}^1_{i1}, \ldots, \tau^n_{i1} \rangle \\ \pi_{i1} : \tau_{i1} \end{array} \right] \ldots \left[\begin{array}{c} \langle \underline{\tau}^1_{ik}, \ldots, \tau^n_{ik} \rangle \\ \pi_{ik} : \tau_{ik} \end{array} \right]$$

wobei die $\mathbf{nt}_j = \langle \tau_{ij}^1, \ldots, \tau_{ij}^n \rangle$ die Nichtterminalsymbole der rechten Seite, und die $\pi_{ij} : \tau_{ij}$ die den Nichtterminalsymbolen zugeordnete Konvertierungsinformation bezeichnen. Tritt die Grammatikregel

$$r : \langle \tau^1, \ldots, \tau^n \rangle \rightarrow \begin{matrix} \langle \sigma_1^1, \ldots, \sigma_1^n \rangle \\ \pi_1 : \rho_1 \end{matrix} \cdots \begin{matrix} \langle \sigma_m^1, \ldots, \sigma_m^n \rangle \\ \pi_m : \rho_m \end{matrix}$$

in einer Ableitung auf, dann ist der Teil der semantischen Repräsentation, der von dieser Regel abgeleitet werden kann, gegeben durch

$$sem(r) = \begin{bmatrix} \theta & \\ \pi_1 & approp(\theta, \pi_1) \sqcup \sigma_1^1 \sqcup \rho_1 \\ \vdots & \vdots \\ \pi_m & approp(\theta, \pi_m) \sqcup \sigma_m^1 \sqcup \rho_m \end{bmatrix} \tag{4.1}$$

wobei θ gegeben ist durch die Unifikation von dem semantischen Typ ρ_1 der Regel r mit allen Typen, die die Merkmalspfade π_i einführen; oder formell:

$$\theta = \tau^1 \sqcup \bigsqcup_{1 \leq i \leq m} intro(\pi_i) \tag{4.2}$$

Dieser Zusammenhang ist in Abbildung 4.2 grapisch veranschaulicht.

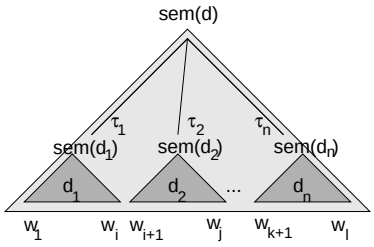


Abbildung 4.2. Eine Ableitung eines Eingabesatzes. Die semantische Repräsentation der gesamten Ableitung ergibt sich aus der semantischen Ableitung der Teilsätze, die als Werte der Merkmalspfade π_i in die Gesamtrepräsentation aufgenommen werden.

Damit die Unifikationen wohldefiniert sind, müssen folgende Bedingungen erfüllt sein:

(i) **Kompatibles Merkmal**
$intro(\pi)$ ist kompatibel zu τ^1

(ii) **Erlaubter Merkmalspfad**
$approp(\tau^1, \pi)$ ist definiert

(iii) **Kompatibler Pfadwert**
$approp(\tau^1, \pi)$ ist kompatibel zu σ^1

Diese Bedingungen können zur Übersetzungszeit der Grammatik überprüft werden. Es kann somit garantiert werden, daß eine gültige Ableitung eine wohlgetypte Merkmalsstruktur ist. Werden allerdings vom Parsingalgorithmus Teile der Eingabe übersprungen (Skip Parser), dann kann nicht mehr garantiert

werden, daß die resultierende Repräsentation wohltypisiert ist. Dies ist ein Mechanismus, um die Probleme mit Spracherkenner-Eingabe zur Laufzeit zu entdecken.

Der vorgeschlagene semantische Konstruktionsalgorithmus macht eine wichtige Annahme. Wenn ein Nichtterminalsymbol $\mathbf{nt}_2$ von einem anderen Nichtterminalsymbol $\mathbf{nt}_1$ dominiert wird, also $\mathbf{nt}_1 \stackrel{*}{\rightarrow} \ldots \mathbf{nt}_2 \ldots$ in einer Ableitung vorkommt, dann ist die semantische Repräsentation ein Teil der semantischen Repräsentation von $\mathbf{nt}_1$, oder formell:

$$sem(\mathbf{nt}_1) = \begin{bmatrix} \sigma \\ f_1 \mid \ldots \mid f_n \; sem(\mathbf{nt}_2) \end{bmatrix}$$

Stellt sich dies als eine Einschränkung heraus, kann durch flache Parsebäume Abhilfe geschaffen werden. Grammatiken für flache Ableitungen tendieren obendrein dazu, sich stabiler in Gegenwart von Erkennungsfehlern zu verhalten.

Label	Deferred Spec.	Instantiated Spec.
1	$\langle obj, N, \_ \rangle$	$\langle obj\_restaurant, N, \_ \rangle$
2	$\langle property, Adj, prd \rangle$	$\langle prp\_nationality, Adj, prd \rangle$
3	$\langle property, Adj, sup \rangle$	$\langle prp\_spatial, Adj, sup \rangle$

Abbildung 4.3. Eine virtuelle Tabelle wird als Zwischenschritt vom Grammatikregel-Übersetzer erzeugt. Die Information in Spalte 1 und 2 stammt von der abstrakten Basisregel. Die Information in Spalte 3 stammt von der Regelspezialisierung.

Abstrakte Basisregeln. In diesem Abschnitt wird eine Technik entwickelt, die es einem Grammatikentwickler ermöglicht, die syntaktischen und semantischen Reprasentationen einer Grammatikregel getrennt zu spezifizieren. Dazu wird die aus dem Bereich der objekt-orientierten Programmiersprachen bekannte Mehrfachvererbung zusammen mit abstrakten Basisklassen auf Grammatikregelspezifikation angewandt.

Insbesondere kann eine strukturelle Grammatikregel *abstrakt* erklärt werden. Eine abstrakte Grammatikregel ist eine strukturelle Grammatikregel, bei der mindestens ein Nichtterminalsymbol der rechten Seite abstrakt deklariert ist. Ein als abstrakt deklariertes Nichtterminalsymbol $\langle \sigma^1, \ldots, \sigma^n \rangle$ ermöglicht es, den semantischen Inhalt dieses Nichtterminalsymbols durch Angabe eines Typs τ zu einem neuen Nichtterminalsymbol $\langle \sigma^1 \sqcup \tau, \ldots, \sigma^n \rangle$ zu spezialisieren. Damit ist der Grammatikschreiber in der Lage, allgemeine Basisgrammatikregeln zu schreiben, die keinerlei domänenspezifische Information enthalten. Diese Basisregeln können in verschiedenen Domänen durch Angabe der domänenspezifischen Information spezialisiert werden.

Das folgende Beispiel erläutert dieses Verfahren. Abbildung 4.4 zeigt drei abstrakte Basisregeln für deutsch, englisch und französisch. Diese Regeln beschreiben die Struktur von Nominalphrasen mit höchstens einem Superlativ und höchstens einem ‘prädikativem Adjektiv. Im Beispiel des Stadtinformationskiosks, diese Regeln beschreiben Satzfragmente wie zum Beispiel *the closest Italian restaurant* oder *le restaurant le plus proche*.

$base_{ger,eng}$: $\langle obj, NP, \_\rangle = 1$
$\rightarrow$ $\langle det, \_, \_\rangle\ \langle property, A, sup\rangle* = 3\ \langle property, A, prd\rangle* = 2\ \langle obj, N, \_\rangle = 1;$

$base_{fr}$: $\langle obj, NP, \_\rangle = 1$
$\rightarrow$ $\langle det, \_, \_\rangle\ \langle obj, N, \_\rangle = 1\quad \langle property, A, prd\rangle* = 2\ \langle property, A, sup\rangle* = 3;$

Abbildung 4.4. Abstrakte Regelspezifikationen für Deutsch, Englisch und Französisch

Abstrakte Nichtterminalsymbole sind gekennzeichnet durch die Angabe eines Indexes der Form $= i$. Der Grammatikregelübersetzer erzeugt eine interne virtuelle Tabelle. Die virtuelle Tabelle ordnet den Indizes der abstrakten Nichtterminalsymbole die semantischen Typen zu. Abbildung 4.5 (a) zeigt die virtuelle Tabelle für die obigen Regeln (für alle drei Regeln wird dieselbe virtuelle Tabelle erzeugt).

NT Index	**Wert**
1	*obj*
2	*property*
3	*property*

(a)

NT Index	**Anwendung 1**	**Anwendung 2**
1	*obj_restaurant*	*obj_flight*
2	*prp_nationality*	*prp_nonstop*
3	*prp_spatial*	*prp_cheap*

(b)

Abbildung 4.5. (a) Die virtuelle Tabelle für die drei abstrakten Basisregeln in Abb. 4.4. (b) Spezialisierungen der abstrakten Basisregeln für zwei verschiedene Anwendungen.

Es genügt dann für die vollständige Instantiierung der abstrakten Basisregeln, die semantischen Typen anzugeben, die die abstrakten Nichtterminalsymbole spezialisieren sollen. Dies geschieht wie folgt:

instantiate $base_{ger,eng,fr}$ with
$obj\_restaurant = 1$
$prp\_nationality = 2$
$prp\_spatial = 3$

Das Ergebis der Instantiierung ist dann äquivalent zu folgenden Grammatikregeln.

$base_{ger,eng}$: $\langle obj\_restaurant, N, \_\rangle$
$\rightarrow$ $\langle det, \_, \_\rangle\ \langle prp\_spatial, A, sup\rangle*\ \{\text{SPATIAL}\ prp\_spatial\}\ \langle prp\_nationality, A, prd\rangle*\ \{\text{NATIONALITY}\ prp\_nationality\}\ \langle obj\_restaurant, N, \_\rangle;$

$base_{fr}$: $\langle obj\_restaurant, N, \_\rangle$
$\rightarrow$ $\langle det, \_, \_\rangle\ \langle obj\_restaurant, N, \_\rangle\ \langle prp\_nationality, A, prd\rangle*\ \{\text{NATIONALITY}\ prp\_nationality\}\ \langle prp\_spatial, A, sup\rangle*\ \{\text{SPATIAL}\ prp\_patial\};$

Abbildung 4.6. Die instantiierten Regeln

Es wird weiterhin bemerkt, daß die semantische Konvertierungsinformation der abstrakten Nichtterminalsymbole erst bei der Instantiierung der abstrakten

Regeln, nicht aber in den abstrakten Regeln selbst angegeben werden kann. Dies liegt daran, daß die Merkmalspfade domänenabhängig sind. Damit sieht die vollständige Instantiierung der Regeln für zwei verschiedene Domänen aus wie in Abbildung 4.7.

instantiate *base* with		instantiate *base* with	
obj_flight	$= 1$,	*obj_restaurant*	$= 1$,
prp_nonstop	$= 2$	*prp_nationality*	$= 2$
{FLIGHTTYPE *prp_nonstop*},		{FLIGHTTYPE *prp_nonstop*},	
prp_price	$= 3$	*prp_spatial*	$= 3$
{PRICE *prp_price*};		{SPATIAL *prp_spatial*};	

Abbildung 4.7. Die Instantiierung der abstrakten Basisregeln für zwei Domänen mit semantischer Konvertierungsinformation

Um zumindest teilweise Konvertierungsinformation in den Basisgrammatikregeln abzulegen, müßte man die Is-Part-Of Relation, die durch die Merkmal angegeben wird, spezialisieren. Dies ist möglich durch die Verwendung ausdrucksstärkerer Beschreibungslogiken. Unter Verwendung von Merkmalshierarchien, die ähnlich den aus den Beschreibungslogiken bekannten Rollenhierarchien (siehe beispielsweise [Horrocks *et al.*, 1999]). Relationen spezialisieren können, könnte man in den Basisregeln abstrakte Relationen angeben. Ein Beispiel ist in Abbildung 4.8 gezeigt.

$$
\begin{array}{lll}
base_{ger,eng} : \langle obj, N, \_\rangle = 1 & & \\
\rightarrow & \langle det, \_, \_\rangle \; \underset{\{\text{PROPERTY} = 4\}}{\langle property, A, sup\rangle *} = 3 & \underset{\{\text{PROPERTY} = 5\}}{\langle property, A, prd\rangle *} = 2 \; \langle obj, N, \_\rangle = 1;
\end{array}
$$

Abbildung 4.8. Abstrakte Basisregel mit abstrakter Konvertierungsinformation.

Mit den Merkmalshierarchien

PROPERTY $\sqsubseteq$ PRP_NATIONALITY
PROPERTY $\sqsubseteq$ PRP_SPATIAL

und

PROPERTY $\sqsubseteq$ PRP_FLIGHTTYPE
PROPERTY $\sqsubseteq$ PRP_PRICE

können Relationen zwischen Typen bereits in den Basisregeln angegeben und bei Regelinstantiierung in geeigneter Weise spezialisiert werden, wie zum Beispiel in

instantiate *base* with	
$\langle obj\_flight, N, \_\rangle$	$= 1$,
$\langle prp\_flighttype, A, prd\rangle$	$= 2$,
$\langle prp\_cheap, A, sup\rangle$	$= 3$,
PRP_FLIGHTTYPE	$= 4$,
PRP_PRICE	$= 5$;

Allerdings erfordern Merkmalshierarchien eine ausdruckstärkere Variante der zugrundeliegenden Merkmalslogik. Wie diese Logik aufgebaut ist, ist der Fokus späterer Arbeit.

4.4 Satzanalyse

In diesem Abschnitt wird auf den verwendeten Parser und die semantische Konstruktion eingegangen.

4.4.1 Robuste Satzanalyse

Der für die Satzanalyse verwendete Parser SOUP [Gavalda, 2000] ist ein einfacher und effizienter Top-Down Parser, der kontextfreie Grammatiken verwendet. Zusätzliche Heuristiken sorgen für Robustheit bei Spracherkennungsfehlern und ungrammatikalischer Eingabe. Da der verwendete Parser vektorisierte Grammatiken nicht verarbeiten kann, werden die vektorisierten Grammatiken zuerst in kontextfreie Grammatiken übersetzt. Hierbei wird ein vektorisiertes Nichtterminalsymbol übersetzt in die Zeichenkette, die es repräsentiert, also zum Beispiel

$$c(\langle \underline{obj\_trainstation}, N, sg \rangle) = obj\_trainstation\$N\$sg$$

wobei c die Übersetzungsfunktion ist.

Eine vektorisierte Grammatikregel wird in eine kontextfreie Regel übersetzt, indem alle vektorisierten Nichtterminalsymbole durch ihre kontextfreien Gegenstücke ersetzt werden.

Bei der Übersetzung der Nichtterminalsymbole in Zeichenketten geht natürlich Information über die partielle Ordnung der Nichtterminalsymbole verloren. Aus diesem Grunde muß die partielle Ordnung explizit in den Grammatikregeln kodiert werden. Mit anderen Worten, wenn $\mathbf{nt}_1 \leq \mathbf{nt}_2$ gilt, dann muß die kontextfreie Grammatikregel

$$c(\mathbf{nt}_1) \rightarrow c(\mathbf{nt}_2)$$

zu der resultierenden Grammatik hinzugefügt werden.

4.4.2 Kompensation des Skip Parsers

Wordskipping des robusten Parsers kann zu inkonsistenten Repräsentationen führen. Abbildung 4.9 zeigt zwei Ableitungen. Die Ableitung in Abbildung 4.9 (a) zeigt die intendierte Struktur des Eingabesatzes; Abbildung 4.9 (b) die tatsächliche Ableitung, hervorgerufen durch das Überspringen von Worten.

Die semantischen Repräsentationen der beiden Ableitungen sehen dann aus wie in Abbildung 4.10.

Das in der Typenhierarchie gespeicherte Wissen kann genutzt werden, um eine solche inkonsistente Repräsentation zu reparieren. Der Algorithmus funktioniert folgendermaßen. Wenn es in der Reprasentation einen Knoten q mit einem Typen gibt, der durch die generalisierte Unifikation zweier Typen $\tau \dot{\sqcup} \sigma$ zustande gekommen ist und gleichzeitig in der Satzableitung an der dem Typen

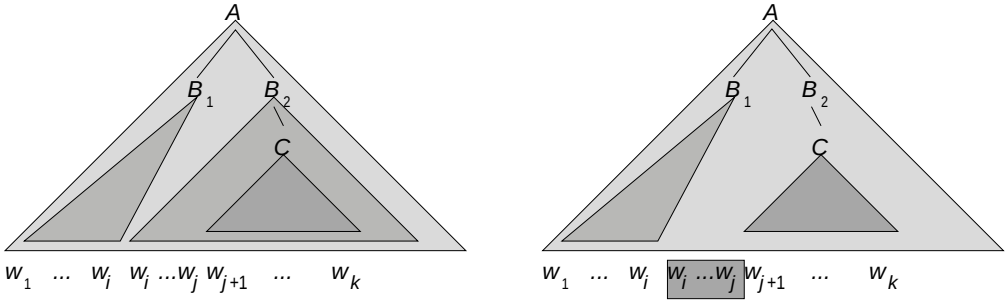


Abbildung 4.9. Zwei Ableitungen für einen Eingabesatz. (a) Die intendierte Ableitung. (b) das Überspringen von Worten führt zu einer anderen Ableitung

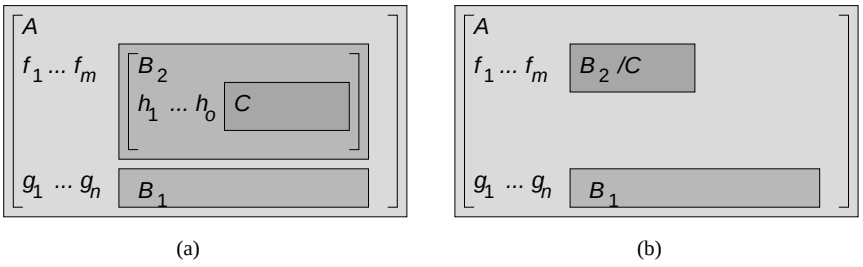


Abbildung 4.10. Die semantischen Repräsentationen der zwei Ableitungen aus Abbildung 4.9. (a) Die intendierte Ableitung. (b) das Überspringen von Worten führt zu einer anderen Ableitung

entsprechenden Stelle Worte übersprungen worden sind, wird untersucht, ob es einen Merkmalspfad $\pi = h_1 \mid \ldots \mid h_o$, so daß $approp(intro(\pi), \pi)$ mit σ kompatibel ist. Wenn dies der Fall ist, wird der Merkmalspfad für den Knoten q definiert, so daß der Wert von π an q C ist. Um den Grammatikschreibern mehr Kontrolle über die Anwendung dieses Algorithmus zu geben, können die Pfade π eingeschränkt werden, so daß der Algorithmus nur an bestimmten Stellen in der Repräsentation zur Anwendung kommt.

4.5 Bestimmung des semantischen Inhalts der Klärungsfragen

In mehreren Fällen wird eine Beschreibung der Objekte unter Diskussion generiert. Dieser Abschnitt diskutiert zielsprach- und domänenunabhängige Verfahren, um den semantischen Inhalt dieser Beschreibungen zu bestimmen.

Als Basis zur Bestimmung von Klärungsfragen dienen die unterspezifizierten Merkmalsstrukturen. Diese wurden so konstruiert, daß Gemeinsamkeiten und Unterschiede zwischen Objektbeschreibungen aus den Repräsentationen hervorgehen. Abbildung 4.11 zeigt ein Beispiel einer unterspezifizierten Merkmalsstruktur, welche sechs Restaurants beschreibt.

$$
\left[\begin{array}{ll}
\multicolumn{2}{l}{obj\_restaurant} \\
\text{NAME} & string^{(1,\ldots,6)}\left\{\begin{array}{l} \text{“}Abruzzi\text{'s } Restaurant\text{“}^{(1)} \\ \text{“}Antonini\ Ristorante\text{“}^{(2)} \\ \text{“}Cozumel\text{“}^{(3)} \\ \text{“}Mad\ Mex\text{“}^{(4,5)} \\ \text{“}Mediterranean\ Grill\text{“}^{(6)} \end{array}\right. \\
\text{ADDR} & \left[\begin{array}{ll} \multicolumn{2}{l}{address} \\ \text{STR-NAME} & string^{(1,\ldots,6)}\left\{\begin{array}{l} \text{“}Forbes\ Ave\text{“}^{(1,3)} \\ \text{“}Jane\ St\text{“}^{(2,4,6)} \\ \text{“}Atwood\ St\text{“}^{(5)} \end{array}\right. \\ \text{STR-NUM} & int^{(1,\ldots,6)}\left\{\begin{array}{l} 52^{(1)} \\ 370^{(2)} \\ 2700^{(3)} \\ 5505^{(4)} \\ 5824^{(5)} \\ 6400^{(6)} \end{array}\right. \end{array}\right] \\
\text{NATIONALITY} & nationality^{(1,\ldots,6)}\left\{\begin{array}{l} prp\_mexican\ ^{(3,4,5)} \\ prp\_greek\ ^{(6)} \\ prp\_italian\ ^{(1,2)} \end{array}\right. \\
\text{SPEC} & spec^{(1,2,4)}\left\{\begin{array}{l} \text{“}patio\text{“}^{(1,2)} \\ \text{“}live\_music\text{“}^{(4)} \end{array}\right.
\end{array}\right]
$$

Abbildung 4.11. Eine unterspezifizierte Merkmalsstruktur, die Restaurants beschreibt. Zwei der Restaurants haben einen Patio, eines bietet Live-Musik an.

Dem Anwender soll nun durch die Angabe von einer Liste von Optionen die Möglichkeit gegeben werden, aus diesen Objektbeschreibungen auszuwählen. Dazu müssen Objektbeschreibungen bestimmt werden, die die repräsentierten Objekte partitionieren. Abbildung 4.12 zeigt Beispiele für Merkmalspfade und die daraus resultierende Partition. Der angegebene Merkmalspfad beschreibt die vom Dialogsystem zu stellende Frage und die dadurch induzierte Partition der Objektmenge. Die zu erwartende Größe der Objektmenge nach Beantwortung der Frage hängt von der Partition ab und ist ebenfalls angegeben.

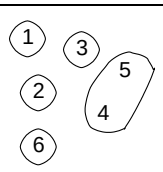
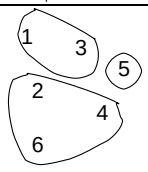
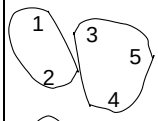
	Partition 1	**Partition 1**	**Partition 3**
Merkmalspfad	NAME	ADDR|STR-NAME	NATIONALITY
Partition			
Mittel	$\frac{6}{5} = 1,2$	$\frac{6}{3} = 2$	$\frac{6}{3} = 2$

Abbildung 4.12. Beispiele für Partitionen der in Abbildung 4.11 gezeigten Objektbeschreibungen. Wird nach dem Namen gefragt, ist die erwartete Größe der verbleibenden Objektmenge 1,2; wird nach Straßenname oder Nationalität gefragt, ist die erwartete Größe der verbleibenden Objektmenge 2.

Um das Beispiel fortzusetzen, nehmen wir an, daß die Klärungsfrage 3 gewählt wird und der Benutzer zwischen mexikanische, griechischen und italienischen Restaurants auswählen kann. Wird mexikanisch gewählt, ergibt sich folgendes Bild.

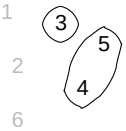
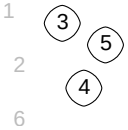
	Partition 1	**Partition 1**
Merkmalspfad	NAME	NAME ADDR|STR-NAME
Partition		
Mittel	$\frac{3}{2} = 1,5$	$\frac{3}{3} = 1$

Abbildung 4.13. Beispiele für Partitionen der in Abbildung 4.11 gezeigten Objektbeschreibungen, nachdem *prp_mexican* gefordert wurde.

Mittels des Namens kann die Objektmenge nicht eindeutig disambiguiert werden. Die zu erwartende Größe der resultierenden Objektmenge ist 1,5. Wird jedoch mittels des Namens und des Straßennamens gleichzeitig disambiguiert, etwa mit der Frage *Would you like to go to Cozumel, to Mad Mex on Atwood Street or to Mad Mex on Jane Street?*, kann eine vollständige Disambiguierung in einem Schritt erreicht werden.

4.6 Generierung Natürlicher Sprache

Nach der Bestimmung des zu vermittelnden Inhalts muß eine Oberflächenform der semantischen Repräsentation erzeugt werden. Dies geschieht mithilfe von *Textschablonen*. Eine Textschablone ist eine Zeichenkette, in der Variablen in Abhängigkeit der Situation ersetzt werden können. Textschablonen sind mit ihrem Typ und einer Menge von Constraints qualifiziert. Die Schablonenqualifikation beschreibt die Dialogsituation, in der die Schablone eingesetzt werden kann. Im folgenden werden Schablonentyp und Constraints sowie die Variablenersetzung genauer beschrieben.

4.6.1 Schablonentypen

Das Dialogsystem unterstützt die Generierung von verschiedenen Typen von Äußerungen.

Definition 3 (Schablonentyp). *Ein* Schablonentyp *charakterisiert den Typ einer vom Dialogsystem zu tätigen Äußerung. Jedem Schablonentyp wird eindeutig ein Sprechakt zugeordnet.*

Die unterstützten Schablonentypen sind in Tabelle 4.1 dargestellt.

Äußerungstyp	**Sprechakt**	**Beispiel**
enumqst	sa_whquestion	Would you like a Chinese, Italian or Mexican restaurant?
infoqst	sa_whquestion	Which neighborhood would you like to go to?
yesnoqst	sa_ynquestion	Do you want to go to Shadyside? Please say yes or no.
illegaldesc	sa_whquestion	I do not know a French restaurant in Squirrel but I do know French restaurants in Shadyside and South Side. Which neighborhood would you like to go to?
statement	sa_statement	I am giving you directions to “Mad Mex“.

Tabelle 4.1. Die vom Dialogsystem unterstützten Äußerungstypen. Ein Äußerungstyp beschreibt eine Schablone genauer als ein Sprechakt. Damit können feinere Abstufungen beider Generierung vorgenommen werden.

4.6.2 Variablen

Eine Variable kann zur Laufzeit durch eine Zeichenkette ersetzt werden. Variablen beziehen sich auf Repräsentationen im Diskurs. Die unterstützten Variablen sind in Tabelle 4.2 aufgelistet.

Variablen referieren auf unterspezifizierte oder partiell unifizierte Merkmalsstrukturen. Daraus ergibt sich die Notwendigkeit, die in der Merkmalsstruktur enthaltenen Elemente einzeln anzusprechen. Dies geschieht durch Zugriffsoperatoren.

Variablenname	Bedeutung
`#sem`	semantische Repräsentation der Äußerung
`#objs`	Repräsentation der Objekte, auf die referiert wird
`#goals`	Dialogziele, die noch aktiv sind
`#last`	Variable, die auf letzte Äußerung des Systems referiert

Tabelle 4.2. Die vom Dialogsystem unterstützten Variablen

Definition 4 (Zugriffsoperatoren). *Die* Zugriffsoperatoren `^first`, `^last`, `^middle` *und* `^all`*, wenn angewandt auf eine Variable V, erlauben es, auf die erste Merkmalsstrukturen, die letzte Merkmalsstruktur oder alle bis auf die erste und die letzte Merkmalsstruktur zuzugreifen, auf die V referiert.*

Das folgende Beispiel soll das Zusammenspiel von Schablonen, Variablen und Zugriffsoperatoren erläutern. Die Eingabe des Anwenders ist der Satz *"Give me directions to a French restaurant in Squirrel Hill"*. Die semantische Repräsentation ist gegeben durch

$$\#sem = \begin{bmatrix} act\_givedirections \\ \text{ARG} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant & \\ \text{NEIGHBORHOOD} & \text{“}Squirrel\ Hill\text{”} \\ \text{NATIONALITY} & prp\_french \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{4.3}$$

In diesem Beispiel ist die Ergebnismenge der Datenbankanfrage leer, da es keine französischen Restaurants im gewünschten Stadtteil gibt. Relaxation der Constraints, wie in Abschnitt 5.3 beschrieben, erzeugt eine Ergebnismenge, die französische Restaurants in anderen Stadtteilen enthält. Die Repräsentation der Ergebnismenge ist gegeben durch die folgende unterspezifizierte Merkmalsstruktur.

$$\#objs = \begin{bmatrix} act\_givedirections^{(1..4)} \\ \text{ARG} \begin{bmatrix} obj\_path^{(1..4)} \\ \text{DST} \begin{bmatrix} obj\_restaurant^{(1..4)} & \\ \text{NEIGHBORHOOD} & string^{(1..4)} \begin{cases} \text{“}Shadyside\text{”}^{(1..3)} \\ \text{“}South\ Side\text{”}^{(2)} \\ \text{“}East\ Liberty\text{”}^{(4)} \end{cases} \\ \text{NATIONALITY} & prp\_french^{(1..4)} \\ \text{ADDRESS} & \ldots \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{4.4}$$

Die folgende Schablone kann dann benutzt werden, um den Anwender über den Inhalt der Ergebnismenge zu informieren.

I do not know a
$\#sem@[\text{ARG}|\text{DST}|\text{NATIONALITY}] :< prp\_nationality : A : \_ >$
$\#sem@[\text{ARG}|\text{DST}] :< obj\_place : N : sg >$ in
$\#sem@[\text{ARG}|\text{DST}|\text{NEIGHBORHOOD}]$.
However, I found
$\#objs\hat{}num$ (4.5)
$\#objs@[\text{ARG}|\text{DST}] :< obj\_place : N : pl >$ in
$\#part\hat{}first@[\text{ARG}|\text{DST}|\text{NEIGHBORHOOD}]$,
$\#part\hat{}middle@[\text{ARG}|\text{DST}|\text{NEIGHBORHOOD}]$ and
$\#part\hat{}last@[\text{ARG}|\text{DST}|\text{NEIGHBORHOOD}]$.
Which neighborhood would you like to go to?

Zunächst wird die Variable $\#sem$ betrachtet. Die Werte der angegebenen Merkmalspfade sind prp_french, $obj\_restaurant$ und "$Squirrel\ \ Hill$", respektive. Ist nach dem Merkmalspfad ein vektorisiertes Nichtterminalsymbol angegeben, wird der Typ des Wertes des Merkmalpfades mit dem ersten Element des vektorisierten Nichtterminalsymbols unifiziert. Da $prp\_nationality \sqsubseteq prp\_french$ und $obj\_place \sqsubseteq obj\_restaurant$ gilt, erhalten wir also die vektorisierten Nichtterminalsymbole $< prp\_french : A : \_ >$ und $< obj\_restaurant : N : sg >$. Unter Verwendung der lexikalischen Grammatikregeln werden die Nichtterminalsymbole durch die Zeichenketten *"French"* und *"restaurant"* ersetzt. Ist kein Nichtterminalsymbol angegeben, wird der Wert des Merkmalpfades unverändert übernommen. Wir erhalten somit den folgenden ersten Satz: *"I do not know a French restaurant in Squirrel Hill."*, wobei die ersetzten Satzteile durch Unterstreichungen kenntlich gemacht worden sind.

Die Variable $\#objs$ bezieht sich auf die in der Datenbank gefundenen Objektrepräsentationen. Damit werden die Ausdrücke $\#objs\hat{}num$ und $\#objs@[\text{ARG}|\text{DST}] :< obj\_place : N : pl >$ durch *4* und *restaurants* ersetzt, respektive. Die Variable $\#part$ bezieht sich auf eine Partition des angegebenen Merkmalpfades. Damit werden die drei folgenden Ausdrücke durch *"Shadyside"*, *"South Side"* und *"East Liberty"* ersetzt. (Wäre statt der Variablen $\#part$ die Variable gewählt worden, würde $\#part\hat{}middle@[\text{ARG}|\text{DST}|\text{NEIGHBORHOOD}]$ durch den Ausdruck *"South Side, Shady Side"* ersetzt worden, was den beiden mittleren Elementen mit den Indices 2 und 3 entspricht. Damit würde die gesamte Aufzählung lauten: *"Shady Side, South Side, Shady Side and Squirrel Hill"*) Damit wird dann folgender Satz generiert: *"However, I found 4 restaurants in Shadyside, South Side and East Liberty. Which neighborhood would you like to go to?"*.

4.6.3 Schablonen-Constraints

Ein Schablonen-Constraint schränkt die Situationen ein, in denen die durch den Constraint partiell beschriebene Schablone eingesetzt werden kann.

Definition 5 (Schablonen-Constraint). *Ein Schablonen-Constraint hat die Form type* : (*const* = *literal*) *wobei literal entweder eine Konstante oder ein Variablenausdruck sein kann.*

Es gibt verschiedene Typen von Schablonen-Constraints. Die Typen charakterisieren, welcher Aspekt der Dialogsituation durch den Constraint eingeschränkt werden soll, und die Constraintgleichung bestimmt, wie der durch den Typ gegebene Aspekt eingeschränkt werden soll. Die folgenden Typen werden unterstützt: $path, less, more$ und $goalstate$.

Pfad Constraints. Ein Schablonen-Constraint vom Typ bestimmt erlaubte Werte eines gegebenen Merkmalspfades. Die erlaubten Konstanten für den Schablonen-Constraint *path* sind $undefined, defined, ambiguous$ und $unique$, mit den offensichtlichen Bedeutungen. Die folgenden Constraints sind gültige Constraints des Typs *path*.

$$path : (ambiguous = \#objs@[\text{ARG}|\text{DST}]) \tag{4.6}$$

$$path : (undefined = \#objs@[\text{ARG}|\text{DST}]) \tag{4.7}$$

Der erste Constraint beschreibt die Situation, in der der Wert des Merkmalspfades ARG|DST mehr als einen Wert annimmt (wie es beispielsweise in (4.4) der Fall ist). Der zweite Constraint beschreibt die Situation, in der der Wert dieses Merkmalpfades undefiniert ist.

Numerische Constraints. Schablonen-Constraints vom Typ $less$ oder $more$ schränken die Anzahl der möglichen Werte von Merkmalspfaden ein. Gültige Konstanten eines numerischen Constraints sind Ganzzahlen; gültige Literale sind Variablenausdrücke. Die Constraints,

$$less : (5 = \#objs@[\text{ARG}|\text{DST}]) \tag{4.8}$$

$$more : (1 = \#objs@[\text{ARG}|\text{DST}]) \tag{4.9}$$

wenn als Konjunktion interpretiert, beschreiben die Situation, in der der Wert des Merkmalspfades ARG|DST zwischen zwei und vier Werte enthält.

Dialogzielzustand. Schablonen-Constraints vom Typ *goalstate* beschreiben die Situation von Dialogzielen (siehe Abschnitt 5.4.1 für die Definition von Dialogzielbeschreibungen). Eine Dialogzielbeschreibung beschreibt eine Aufgabe, die die Anwendung ausführen kann, wie beispielsweise Fahrtanweisungen zu geben. Die Dialogzielbeschreibung ist im Zustand *selected*, wenn alle anderen Beschreibungen ausgeschlossen wurden, und im Zustand $finalized$ wenn der der Dialogzielbeschreibung zugeordnete Dienst ausgeführt werden kann. Damit sind *selected* und $finalized$ gültige Konstanten und die Namen der Dialogzielbeschreibungen sind gültige Literale. Der Constraint

$$goalstate : (finalized = Goal\_GiveDirections) \tag{4.10}$$

beispielsweise ist erfüllt, wenn die Dialogzielbeschreibung mit dem Namen $Goal\_GiveDirections$ ausgeführt wird.

4.7 Diskussion

Die in diesem Kapitel vorgestellten Mechanismen zur Grammatikverarbeitung sind symbolischer Natur. In verschiedenen Arbeiten wurden Ansätze verfolgt, Grammatiken zu lernen [Walker *et al.*, 2001,Buo, 1996]. Der Vorteil von lernenden Systemen ist, das der Aufwand zum Schreiben von Grammatik- und Konvertierungsregeln reduziert wird. Auf der anderen Seite muß eine entsprechend große Menge von Beispielsätzen vorliegen. Da in dieser Arbeit der schnelle Entwurf von Systemen im Vordergrund stand, kann nicht von ausreichend vielen Datenmengen ausgegangen werden. Der Nachteil symbolischer Ansätze, der hohe Entwicklungsaufwand, wurde durch die Vererbung von Grammatikregeln und durch das Typisieren der Konvertierungsregeln gemildert. Damit können existierende Komponenten wiederverwendet werden und Inkonsistenzen in den Spezifikationen schnell entdeckt werden.

In [Rayner *et al.*, 2001] wurde ein einfacher Ansatz zur Modularisierung von Grammatikregeln beschrieben. Diese Arbeit geht jedoch nicht soweit wie die vorliegende; insbesondere sind keine Vererbungsmechanismen, keine Konvertierungsregeln in semantischen Repräsentationen und keine Typüberprüfungen vorgesehen.

4.8 Zusammenfassung

In diesem Kapitel wurden mehrere für generische Dialogverarbeitung notwendigen Verfahren beschrieben. Die für die Algorithmen nötigen Spezifikationen können in unterschiedlichen Namensräumen spezifiziert werden. Somit können semantisch zusammenhängende Spezifikationen in eigenständigen, wiederverwendbaren Modulen erfolgen.

Die beschriebenen Grammatikformalismen erlauben die Separierung von syntaktischen und semantischen Aspekten. Da sich in vielen aufgabenorientierten Dialogsystemen die syntaktische Form der Anfragen ähneln, können somit Teile der Grammatiken wiederverwendet werden.

Der Satzanalysealgorithmus weist den Eingabesätzen Struktur zu und erzeugt mithilfe der Grammatikannotationen die semantische Repräsentation. Eine wesentliche Eigenschaft des hier vorgestellten Verfahrens ist, daß die semantischen Repräsentationen typisiert sind. Somit kann sichergestellt werden, daß sich die verschiedenen semantischen Aspekte der Eingabe an “derselben Stelle“ in der semantischen Repräsentation wiederfinden.

Allgemeine Algorithmen können auf der Basis von unterspezifizierten Merkmalsstrukturen den semantischen Inhalt von Klärungsfragen bestimmen. Auf dem Ergebnis dieses Verfahrens aufbauend, werden geeignete Schablonen ausgewählt, um eine Oberflächenform der Klärungsfrage zu erhalten.

Alle hier angegebenen Algorithmen sind domänen- und zielsprachunabhängig; sie werden lediglich durch die verschiedenen Spezifikationen (Ontologie, Grammatik und Schablonen-Constraints) gesteuert.

Kapitel 5

Diskurs, Dialog und Datenbankdienste

Die vorausgehenden Kapitel beschäftigten sich mit semantischen Repräsentationen und der Transformation von Text in semantische Repräsentation und umgekehrt. Die eigentliche Dialogverarbeitung beginnt allerdings erst nachdem die semantische Repräsentation einer Äußerung generiert worden ist. Ein Dialogsystem muß beispielsweise bestimmen, auf welche Objekte definite Beschreibungen referieren, ob die Intention des Benutzer eindeutig erkannt werden konnte, und wie die semantische Äußerungen im Diskurs verankert (engl. *grounding*) werden müssen. Dieses Kapitel beschäftigt sich mit Diensten, die diese Funktionen implementieren. Genau wie die im letzten Kapitel beschriebenen Dienste wird die Entscheidung, welche Dienst zu welcher Zeit angefordert wird, in der *Benutzerschnittstellenschicht* getroffen.

5.1 Einleitung

Nachdem der Eingabetext, wie im letzten Kapitel beschrieben, analysiert und in eine semantische Repräsentation umgewandelt worden ist, muß das Dialogsystem entscheiden, was als nächstes zu tun ist. Es ist das Ziel des Benutzers, einen von der Anwendung angebotenen Dienst anzufordern, wie zum Beispiel ein Hotel zu reservieren oder einen Flug zu buchen. Dies ist vergleichbar mit den von graphischen Oberflächen bekannten *callback* Mechanismen [Myers, 1991] oder typisierten Ereignismechanismen wie sie von JAVA her bekannt sind. Was graphische Benutzeroberflächen von sprachgesteuerten Oberflächen unterscheidet, ist, daß sprachgesteuerte Oberflächen erlauben, den beabsichtigten Dienst der Anwendung nur partiell zu spezifizieren. Wenn beispielsweise die Äußerung des Benutzers ist *"Buche einen Flug"*, dann muß durch nachfolgende Interaktion mit dem Benutzer und der Datenbank fehlende Information über das Objekt unter Diskussion (der Flug) solange eingeholt werden, bis genügend Information vorhanden ist, um den entsprechenden Dienst der Anwendung anzufordern.

Konsequenterweise muß es dann einen *informationsbasierten* oder *datengetriebenen* Mechanismus geben, der dem Dialogsystem anzeigt, wann der Dialog mit dem Benutzer genügend Information erbracht hat, so daß eine Datenbankanfrage generiert oder ein Dienst der Anwendung angefordert werden kann.

Dieser eher indirekt agierende Mechanismus steht in Kontrast zu den eher ereignisbasierten Interaktionsmechanismen graphischer Benutzeroberflächen. In graphischen Benutzerschnittstellen gibt es eine eins-zu-eins Beziehung zwischen Ereignissen und Aktion; so werden beispielsweise durch einen Mausklick auf einen Knopf alle im Knopf registrierten *Listener* von dem Knopfdruck informiert. In natürlichsprachlichen Benutzerschnittstellen gibt es jedoch im wesentlichen nur ein Ereignis; nämlich, daß der Benutzer eine Äußerung produziert hat. Erst nach der semantischen Analyse der Äußerung kann das Dialogsystem entscheiden, wie es zu agieren hat.

In diesem Kapitel wird der Zusammenhang zwischen der im Diskurs enthaltenen Information und Anforderung von Diensten der Anwendung beschrieben. Hier sind im besonderen drei Aspekte wichtig. Zum ersten gilt es die Frage zu beantworten, wann und wie die semantischen Repräsentation im Diskurs verankert werden sollen. Dies ist eine Interaktion zwischen Komponenten innerhalb des Dialogsystems und nicht zwischen Dialogsystem und Anwendung.

Zweitens muß spezifiziert werden, wann im Diskurs genügend spezifische Objektbeschreibungen vorliegen, so daß eine Datenbankanfrage effizient durchgeführt werden kann. Dies bezieht sich auf definite Beschreibungen (*"das nächste Restaurant*) genauso wie auf Eigennamen (*"zum Bundesverfassungsgericht"*). In diesem Zusammenhang wird Datenbankanfrage sehr weitläufig verstanden. Eine Datenbankanfrage ist der Aufruf eines Dienstes durch eine Objektbeschreibung, der eine Menge von Objektbeschreibungen zurückliefert. Dieser Dienst kann natürlich durch SQL Datenbanken, aber auch durch Internet-Suchmaschinen oder sogar Dienste, die Referenzen auf passende JAVA Objekte liefern, implementiert werden.

Der dritte Aspekt ist es, für das Dialogsystem zu bestimmen, wann die von Benutzer intendierte Aktion der Anwendung eindeutig bestimmt werden kann. Da sich die vorliegende Arbeit auf aufgabenorientierte Dialogsysteme beschränkt, kann die Bestimmung der Intention des Benutzers durch die Auswahl von einer aus einer Menge von vorher bestimmten Aktionen realisiert werden. Dies geschieht durch das sukzessive Entfernen von den Aktionsbeschreibungen aus einer Kandidatenliste, die inkompatibel mit der Information im Diskurs sind. Wenn die Kandidatenliste nur noch eine Aktionsbeschreibung enthält, und alle zur Ausführung des der Beschreibung zugeordneten Dienstes notwendige Information akquiriert worden ist, kann der entsprechende Dienst angefordert werden (oder informell ausgedrückt: das Dialogziel wurde erreicht). Darüberhinaus wird definiert, wenn den Dialog unterstützende Dienste angefordert werden können bevor das Dialogziel erreicht worden ist.

Es sei bemerkt, daß ein vierter Mechanismus zur informationsbasierten Interaktion zwischen Komponenten bereits im Abschnitt 2.5.3 über objekt-orientierte Merkmalsstrukturen definiert worden ist. Die objekt-orientierte Erweiterung, die es ermöglicht, Informationen von externen Wissensquellen in die Merkmalsstrukturen zu inkorporieren, ist lediglich eine Alternative zur Beschreibung datengetriebener Interaktion zwischen Komponenten. Im Gegensatz zu den in diesem Kapitel beschriebenen Mechanismen speichern objekt-orientierte Merkmalsstrukturen keine Zustandsinformation außer der bis zum gegenwärtigen Zeitpunkt angeforderten Dienste; diese Zustandsrepräsentation

ist lokal in den Merkmalsstrukturen abgelegt. Die hier beschriebenen Algorithmen basieren hingegen auf externen Zustandsrepräsentationen wie beispielsweise Diskursinformation.

5.2 Diskursbezogene Dienste

In Dialogsystemen muß eine geeignete Repräsentation des Diskurses vorliegen, damit der Dialog vom Dialogsystem korrekt vorangetrieben und die angemessenen Dienstanforderungen ausgeführt werden können. In frühen Dialogsystemen wurde Diskurs durch eine verkettete Liste von Rahmen repräsentiert. Baumartige Repräsentationen sind jedoch als adäquate Struktur zur Diskursrepräsentation bekannt [Grosz and Sidner, 1986]. Im vorliegenden System ist eine baumartige Repräsentation des Diskurses implementiert. Innere Knoten des Baumes haben keinen Inhalt, sondern verweisen auf die Töchterbäume. Ein Blatt im Diskursbaum repräsentiert einen Turn und enthält Repräsentationen für die Äußerung selbst in Textform, die semantische Repräsentation der Äußerung sowie gegebenenfalls Repräsentationen der Objekte, auf die in der Äußerung referiert wird. Zusätzlich gibt einen Verweise auf den aktuellen Knoten im Baum. Abbildung 5.1 zeigt einen Abbildung für eine Diskursrepräsentation.

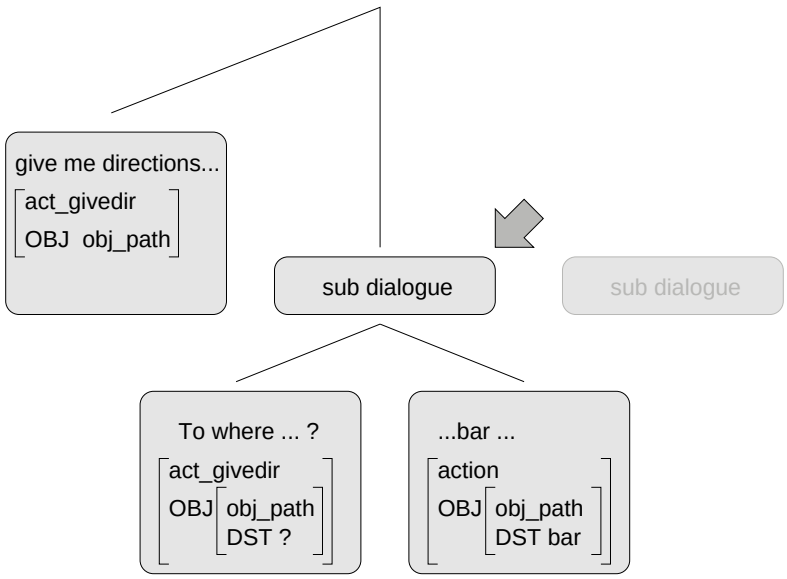


Abbildung 5.1. Beispiel einer baumartigen Diskursrepräsentation. Innere Knoten des Baumes verweisen auf Unterdialoge. Blätter enthalten Repräsentationen des Turns, dessen semantischer Repräsentation und der referenzierten Objektbeschreibungen. Der Pfeil zeigt auf den Knoten, der gegenwärtig unter Diskussion steht. Da der Unterdialog angeschlossen ist, wird der Mutterknoten des abgeschlossenen Dialogs als der aktuelle Knoten betrachtet. Der schattiert gezeichnete Knoten deutet die Position eines möglicherweise folgenden Unterdialogs an.

Um die Diskursrepräsentation anpassen zu können, können die folgenden Operationen auf dem Baum ausgeführt werden. Die Operation $hinzufügen()$ fügt einen neuen Blattknoten als Schwesterknoten des gegenwärtig aktuellen

Knoten in den Diskurs ein und verweist auf den neuen Knoten als den aktuellen Knoten. Diese Operation wird beispielsweise verwandt, um eine Antwort auf eine Frage in den Diskurs einzufügen. Die Operation *erzeugen*() fügt einen neuen inneren Knoten als Schwesterknoten des gegenwärtig aktuellen Knoten in den Diskurs ein und verweist auf den neuen Knoten als den aktuellen Knoten. Mit dieser Operation werden beispielsweise neue Subdialoge eingeleitet. Die Operation *abschließen*() setzt den Mutterknoten des aktuellen Knotens als neuen aktuellen Knoten. So wird beispielsweise das Beenden von Unterdialogen verarbeitet. Abbildung 5.2 stellt die Operationen auf dem Diskursbaum graphisch dar.

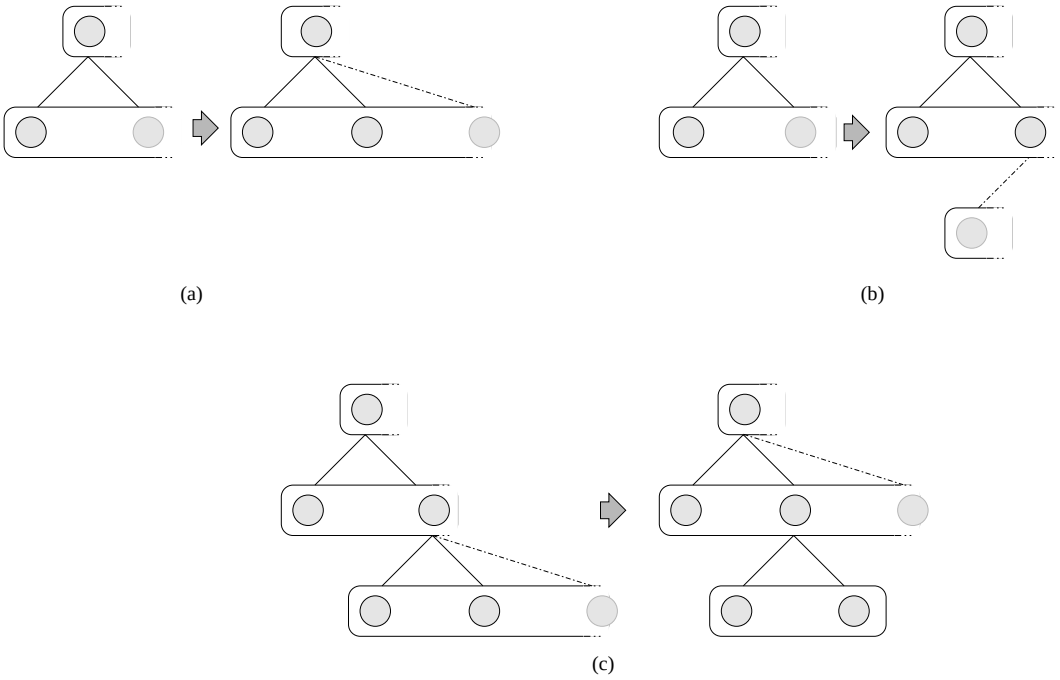


Abbildung 5.2. Beispiel von Operationen auf der Diskursrepräsentation. Jeder Kreis entspricht einem Turn. Die leicht gezeichneten Kreise zeigen die Stelle der nächsten Äußerung an. (a) Einfügen eines neuen Turns. (b) Erzeugen eines Subdialogs. (c) Abschließen eines Subdialogs.

5.3 Referierende Ausdrücke und Datenbankanfragen

Natürlichsprachliche Dialogsysteme greifen typischerweise auf die eine oder andere Form relationaler Datenbanken zu, in der Beschreibungen der Objekte in der Domäne abgelegt sind. Im Falle der Atis Systeme beispielsweise besteht die Datenbank aus Information über Flughäfen, Flugverbindungen, Flugzeuge und so weiter. Im Stadtinformationssystem bestehen die Datenbanken aus Kartendaten, Informationen über Restaurants, Kinos oder Bars. Benutzer können auf Datenbankinformation entweder explizit referieren (*"Was ist der kürzeste Weg zur Fakultät für Informatik?"*) oder die Objekte durch ihre Eigenschaften identifizieren (*"Show me the first flight from Pittsburgh to Boston in the morning*

on USAir“). Es ist Aufgabe der hier beschriebenen Datenbankdienste, die vom Benutzer beschriebenen Objekte aus der Datenbank anzufragen, oder, im Falle daß diese Objekte nicht existieren, ähnliche Objekte zu finden.

Definition 1 (Datenbankanfrage). *Eine* Datenbankanfrage *ist gegeben durch eine Merkmalsstruktur F. Das* Ergebnis einer Datenbankanfrage *ist eine möglicherweise leere Menge von Merkmalsstrukturen $\{F_1, \ldots, F_n\}$.*

Im allgemeinen finden Datenbankanfragen statt, nachdem die semantische Repräsentation einer Äußerung in den Diskurs eingefügt worden ist (aber siehe Abschnitt 5.3.2 für eine feinere Kontrolle des Datenbankzugriffes). Abbildung 5.3 stellt die Beziehung zwischen Datenbankzugriff und Diskurs her.

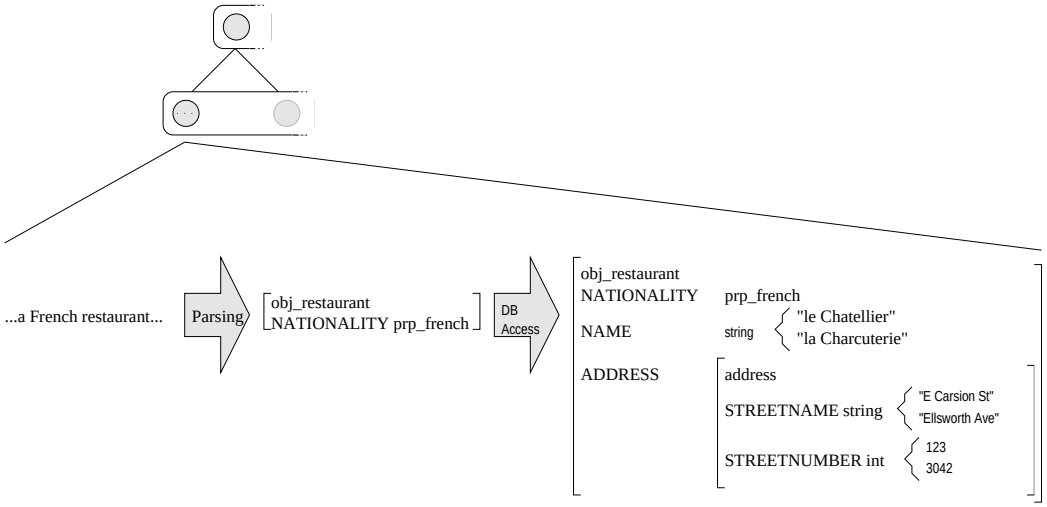


Abbildung 5.3. Zusammenhang zwischen Datenbankzugriff und Diskursbaum. In einem Knoten des Diskursbaumes (dargestellt durch einen Kreis) sind die drei Repräsentationsebenen Test, semantische Repräsentation und Objektrepräsentation enthalten.

5.3.1 Datenbank-Konvertierungsbeschreibungen

Da die Datenbankanfrage selbst durch eine Merkmalsstruktur F gegeben ist, muß die Information in F in eine der Datenbank verständliche Anfrage umgewandelt werden. Dies geschieht unter Verwendung von Transformationsregeln durch einen sprach- und domänenunabhängigen Algorithmus. Im folgenden wird dieser Algorithmus am Beispiel von SQL Datenbanken beschrieben; es ist jedoch einfach möglich, Anfragen für Datenbanken anderer Formate zu implementieren. Die Transformationsregeln übersetzen eine Merkmalsstruktur in eine Datenbankanfrage und konvertieren, durch inverse Anwendung, die Ergebnismenge der Datenbankanfrage in eine Menge von Merkmalsstrukturen.

Die Tabellen in einer SQL Datenbank sind insofern *selbstbeschreibend* als daß die Felder und Datentypen der Felder angefragt werden können. Um eine Merkmalsstruktur in eine Datenbankanfrage umwandeln zu können, muß

der Systementwickler Beziehungen zwischen den Feldern in den Tabellen der Datenbank und Merkmalspfaden in den Merkmalsstrukturen angeben. Die angegebenen Beziehungen können (unter Verwendung der Typenhierarchie und den selbstbeschreibenden Tabellen) auf Typsicherheit überprüft werden, so daß sichergestellt werden kann, daß das Ergebnis einer Datenbankanfrage nur wohltypisierte Merkmalsstrukturen erzeugt.

Sei im folgenden DB eine relationale Datenbank mit Tabellen $T = \{t_1, \ldots, t_n\}$ sowie Verknüpfungen $L \subseteq T \times T$ zwischen den Tabellen, die über Primärschlüssel $\{l_{ij} : 1 \leq i, j \leq n, i \neq j\}$ definiert sind. Mit jeder Tabelle t ist die Menge der Felder $f(t)$ assoziiert, die t definiert sind. Jeder Tabelle t wird ein Typ $\theta(t)$ der Typenhierarchie $\langle \mathsf{Type}, \sqsubseteq \rangle$ zugewiesen, der beschreibt, welche Repräsentationen in der Tabelle t angefragt werden können. Damit können nun die Konvertierungsregeln für Datenbankeinträge definiert werden.

Definition 2 (Datenbankkonvertierungsregeln). *Die Konvertierungsregeln $r(t)$ für eine Tabelle t einer Datenbank bestehen aus Paaren $\langle \pi : \tau, f \rangle$ wobei π ein Merkmalspfad und τ ein Typ ist, so daß $approp(\theta(t), \pi)$ wohldefiniert ist, und f das zu konvertierende Feld in der Datenbank bezeichnet.*

Wir nehmen im folgenden an, daß die Datenbank nur aus einer Tabelle t_1 mit $f(t_1) = \{f_1, \ldots, f_{n_1}\}$ besteht. Weiterhin habe die Merkmalsstruktur F, die die Anfrage repräsentiert, k Merkmalspfade π_i definiert, für die die entsprechenden Konvertierungsregeln $\langle \pi_i : \tau_i, f_i \rangle$ vorhanden sind. Schließlich nehmen wir an, daß die Funktion $val : \mathsf{Type} \rightarrow \mathit{String}$ Typen in Zeichenketten konvertiert (die Funktion ist unten genauer beschrieben). Die Typen der Werte der Merkmalspfade werden mit v_i bezeichnet. Dann ist die SQL Anfrage von folgender Form.

SELECT				
$t_1.f_1, \cdots, t_1.f_{n_1}$				Alle Felder f_i auswählen
FROM t_1				die in t_1 vorhanden sind
WHERE				wobei
$t_1.f_i$	$=$	$val(v_1)$	AND	die Füller der Felder den
$\ldots$		$\ldots$	AND	in der Merkmalsstruktur F
$t_1.f_k$	$=$	$val(v_k)$		angegebenen Typen entsprechen

(5.1)

Die Generierung der SQL Anfrage gestaltet sich komplizierter, wenn mehrere Verknüpfungen in der Datenbank vorhanden sind. Hier wird zunächst durch Inspektion aller bekannten Konvertierungsregeln die Menge $\tilde{T} \subseteq T$ der in die Anfrage involvierten Tabellen $t_1, \ldots, t_m$ bestimmt. Dann definiert $\tilde{T}$ die Knoten und $\tilde{L} = (\tilde{T} \times \tilde{T}) \cap L$ die Kanten eines Graphs, der alle für die Anfrage benötigten Tabellen und eine Obermenge der benötigten Verknüpfungen enthält. Ein Minimum-Spanning-Tree Algorithmus über dem Graphen bestimmt anschließend die Menge $E \subseteq \tilde{T}$ aller zur Anfrage benötigten Verknüpfungen. Die SQL Anfrage besteht dann aus zwei Teilen: Der erste Teil enthält für jede Tabelle

t_i eine Anfrage analog zu 5.2, und der zweite Teil besteht aus Constraints, der die involvierten Tabellen miteinander verknüpft.

$$
\begin{array}{lll}
\textsc{Select} & & \\
\quad t_1.f_1, & \cdots, t_1.f_{n_1} & \text{Alle Felder } f_i \text{ auswählen} \\
\quad \vdots & \vdots & \text{aus allen beteiligten Tabellen} \\
\quad t_m.f_1, & \cdots, t_m.f_{n_m} & \text{auswählen} \\
\textsc{From}\ t_1, \ldots, t_m & & \\
\textsc{Where} & & \text{wobei} \\
\quad t_1.f_{i_1^1} = val(v_1)\ \textsc{And} & & \\
\quad \vdots \qquad\qquad \vdots & & \text{Constraints der ersten Tabelle} \\
\quad t_1.f_{i_{k_1}^1} = val(v_o)\ \textsc{And} & & \\
\quad \vdots \qquad\qquad \vdots & & \\
\quad t_m.f_{i_1^m} = val(v_p)\ \textsc{And} & & \\
\quad \vdots \qquad\qquad \vdots & & \text{Constraints der } m\text{-ten Tabelle} \\
\quad t_m.f_{i_{k_m}^m} = val(v_k)\ \textsc{And} & & \\
\quad t_i.f_j = t_k.f_l & \forall \langle t_i.f_j, t_k.f_l \rangle \in E \rangle &
\end{array}
\tag{5.2}
$$

Die Konvertierung *val* eines Typs in einen String verdient besondere Beachtung. Typisierte Merkmalsstrukturen erlauben die Repräsentation partieller Information durch die Angabe nicht maximaler Typen. Die folgenden Merkmalsstruktur beispielsweise

$$\begin{bmatrix} \textit{obj\_restaurant} \\ \textsc{nationality}\ \textit{asian} \end{bmatrix}$$

subsumiert die beiden folgenden maximalen Merkmalsstrukturen

$$\begin{bmatrix} \textit{obj\_restaurant} \\ \textsc{nationality}\ \textit{japanese} \end{bmatrix} \begin{bmatrix} \textit{obj\_restaurant} \\ \textsc{nationality}\ \textit{chinese} \end{bmatrix}$$

wenn wir $\textit{asian} \sqsubset \textit{japanese}, \textit{chinese}$ annehmen. Allerdings ist Zugriff auf die Subsumptionsbeziehung von Seiten der Datenbank aus nicht möglich, da die Füller der Felder alle als Strings behandelt werden.[1] Dies bedeutet, daß eine Sql Anfrage der Form

$$
\begin{array}{ll}
\textsc{Select} & \\
\quad \textit{Restaurants.name}, \textit{Restaurants.nationality} & \\
\textsc{From}\ \textit{Restaurants} & \\
\textsc{Where} & \\
\quad \textit{Restaurant.nationality} & = \textit{asian}
\end{array}
$$

nicht in der Lage ist, Objekte zu finden, deren NATIONALITY Feld mit dem Wert *chinese* oder *japanese* gefüllt ist. Deswegen definieren wir die Konvertierung von Typen wie folgt.

[1] Diese Problem könnte durch Verwendung einer objekt-orientierten Datenbank umgangen werden. Allerdings sind objekt-orientierte Datenbanksysteme noch nicht weitläufig akzeptiert.

Definition 3 (Typen-Konversion). *Die* Typenkonversion $val(\tau)$ *eines Typs* $\tau \in \langle \mathsf{Type}, \sqsubseteq \rangle$ *in eine Zeichenkette ist definiert als die Zeichenkette "*τ_1 OR ... OR τ_m*" wobei* $\{\tau_1, \ldots, \tau_m\}$ *die Menge aller maximalen Typen ist, die von* τ *subsumiert werden.*

Die Definition impliziert, daß $val(\tau) =$ "τ" wenn immer τ maximal ist. Im obigen Beispiel würde die Datenbankanfrage wie folgt aussehen.

```
SELECT
    Restaurants.name, Restaurants.nationality
FROM Restaurants
WHERE
    Restaurant.nationality      = japanese   OR
    Restaurant.nationality      = chinese    OR
    Restaurant.nationality      = thai       OR
    Restaurant.nationality      = korean
```

falls *japanese, chinese, thai* und *korean* alle der Typenhierarchie asiatische Nationalitäten sind.

5.3.2 Steuerung des Datenbankzugriffes

Die bis jetzt angegebene Konvertierungsinformation erlaubt es dem Dialogsystem, eine partiell spezifizierte Merkmalsstruktur in eine SQL Anfrage umzuwandeln. Die Elemente der Ergebnismenge werden auf umgekehrten Weg zu einer unterspezifizierten Merkmalsstruktur zurückgewandelt. Die unterspezifizierte Struktur dient dann als Basis zur Generierung von Klärungsfragen, wie in Abschnitt 4.5 beschrieben. Dieses Verfahren ist deswegen vorteilhaft, weil dem Benutzer nur die Objekte zur Auswahl angeboten werden, die tatsächlich in der Datenbank vorhanden sind.

Manchmal ist es allerdings besser, Datenbankzugriffe zu einem späteren Zeitpunkt durchzuführen. Inbesondere große Datenbanken führen in Kombination mit wenig informativen Merkmalsstrukturen F zu großen Ergebnismengen. Der Satz *"Show me all flights"* in der ATIS Domäne würde alle Flüge der USA als Ergebnismenge zurückliefern und entsprechend zeitaufwendig zu verarbeiten sein.

Um diese Situation zu vermeiden, können sogenannte *Datenbankwächter* spezifiziert werden.

Definition 4 (Datenbankwächter). *Ein* Wächter G *für eine Datenbank* D *ist eine typisierte Merkmalsstruktur, die den Zugriff auf* D *kontrolliert. Nur die Merkmalsstrukturen* F *werden als Basis für Datenanfragen erlaubt, die von* G *subsumiert werden.*

Als ein Beispiel soll der Zugriff auf die Restaurant-Datenbank beschränkt werden. Ein Datenbankzugriff soll nur dann erfolgen, wenn mindestens die Nationalität, die Preiskategorie oder der Name des Restaurants bekannt ist. Die Datenbankwächter sehen aus wie folgt.

Datenbankwächter

```
guard Restaurant1 {
 [NATIONALITY] >  prp_nationality,
 []            >= obj_restaurant
};

guard Restaurant2 {
 [PRICE] >  prp_price,
 []      >= obj_restaurant
};

guard Restaurant3 {
 [NAME] >  string,
 []     >= obj_restaurant
};
```

Abbildung 5.4. Ein Beispiel für Datenbankwächter

5.4 Dialogziele

Dialogsysteme verfügen typischerweise über ein Modell, das beschreibt, welche Aufgaben das System lösen kann (engl. *task model*) [Flycht-Eriksson, 1999]. Die Komplexität des Aufgabenmodells ist offensichtlich abhängig von der Komplexität der zu lösenden Aufgabe. Manche Dialogsysteme verfügen über KI-Planungssysteme, wie zum Beispiel TRAINS[Allen *et al.*, 1996], während hingegen die meisten Systeme eine Art rahmenbasiertes Aufgabenmodell (engl. *form filling*) implementieren [Papineni *et al.*, 1999,Ward, 1994]. Das Aufgabenmodell stellt das Verbindungselemente zwischen der Sprachverarbeitungskomponente und der eigentlichen Anwendung dar.

Im dem hier verfolgten Ansatz wurde ein einfaches Aufgabenmodell gewählt. Das Aufgabenmodell besteht aus einer Menge von Dialogzielbeschreibungen. Jedes Dialogziel beschreibt einen Dienst, den die Anwendung ausführen kann. Einem Dialogziel ist eine Merkmalsstruktur zugeordnet, die den minimalen Informationsgehalt beschreibt, der zum Ausführen des assoziierten Dienstes notwendig ist. Diese Art des Aufgabenmodells ist eine erweiterte Form des in [Denecke and Waibel, 1997] beschriebenen und ist ähnlich de in [Papineni *et al.*, 1999] beschriebenen *Forms*.

Der Grund für die Wahl eines einfachen Aufgabenmodells ist (wieder einmal) die Anforderung an das Dialogsystem, domänenunabhängig zu sein. Die Überlegung ist wie folgt. Wenn eine bestimmte Anwendung ein komplexeres Aufgabenmodell benötigt, dann muß es die Funktion eines Moduls außerhalb des Dialogsystems sein, diese Funktionalität zu implementieren. Das gewählte Aufgabenmodell hat somit lediglich die Funktion, externe Module von der Notwendigkeit einer Dienstanforderung zu informieren. Damit bleibt das Aufgabenmodell einfach und domänenunabhängig.

5.4.1 Dialogzielbeschreibungen

Eine Dialogzielbeschreibung ordnet eine minimale Menge an Informationen einem Dienst der Anwendung zu. Die Informationsmenge ist beschrieben durch

eine Merkmalsstruktur. Die Bedeutung dieser Zuordnung ist, daß, wenn immer die von der Dialogzielbeschreibung geforderte Informationsmenge im Diskurs vorhanden ist, und darüber hinaus alle anderen Dialogzielbeschreibungen inkompatibel mit der Information mit Diskurs sind, wurde die Intention des Benutzers vom Dialogsystem erkannt, und der der Dialogzielbeschreibung zugeordnete Dienst kann angefordert werden. Abbildung 5.5 zeigt ein Beispiel für eine Dialogzielbeschreibung.

$$\begin{bmatrix} \textit{act\_givedirections} \\ \text{ARG} \begin{bmatrix} \textit{obj\_building} \\ \text{NAME } \textit{string} \\ \text{POSX } \textit{int} \\ \text{POSY } \textit{int} \end{bmatrix} \end{bmatrix}$$

Abbildung 5.5. Eine Dialogzielbeschreibung mit einer unteren informationellen Schranke. Um Fahrtanweisungen geben zu können, muß der Name des Zielortes und dessen Position bekannt sein.

Zusätzlich ist es möglich, Defaultinformation in der Dialogzielbeschreibung abzulegen, um die Dialoge zu verkürzen. In Abbildung 5.6 ist eine Dialogzielbeschreibung gezeigt, die als Abflugsort Pittsburgh annimmt, wenn keine gegensätzliche Information im Diskurs vorliegt.

$$\begin{bmatrix} \textit{act\_givedirections} \\ \text{ARG} \begin{bmatrix} \textit{obj\_home} \\ \text{NAME } \textit{“My home“} \\ \text{POSX } \textit{234} \\ \text{POSY } \textit{256} \end{bmatrix} \end{bmatrix}$$

Abbildung 5.6. Default-Information. Ist kein Zielort angegeben, wird nach Hause gefahren

Darüber hinaus ist es möglich, die Anzahl der Objekte im Diskurs zu beschränken, die eine Dialogzielbeschreibung erfüllen. Wenn ein Benutzer aus einer Menge möglicher Flüge auswählen möchte, könnte zum Beispiel die Anzahl der die Beschreibung erfüllenden Objekte mit der Bildschirmgröße korrelieren. Wenn der Benutzer hingegen einen Flug buchen möchte, muß das den Flug beschriebende Objekt eindeutig bestimmt sein.

Als weitere Interaktion mit der Anwendung besteht die Möglichkeit, Dienste anzufordern, bevor das vom Benutzer intendierte Dialogziel feststeht. Wenn in einem Stadtinformationskiosks beispielsweise eine Hotelreservierung vorgenommen werden soll, sollten die verschiedenen Hotels bereits auf einer Karte angezeigt werden (ein Dienst der Anwendung), bevor das eigentliche Dialogziel, nämlich eine Hotelreservierung durchzuführen, erreicht ist. Abbildung 5.7 zeigt ein solches Beispiel.

$$\begin{bmatrix} act\_givedirections \\ \text{ARG} \begin{bmatrix} obj\_hotel \\ \text{POSX } int \\ \text{POSY } int \end{bmatrix} \end{bmatrix}$$

Abbildung 5.7. Unterziele. Sind die Positionen von Hotels bekannt, werden sie auf der Karte angezeigt, auch wenn das zu reservierende Hotel noch nicht eindeutig bestimmt worden ist.

Schließlich muß die Dialogzielbeschreibung noch die Dienstanforderungen der Dienst enthalten, die bei Erreichen des Dialogzieles aufgerufen werden. Dies wird erreicht durch die Angabe der Dienste der Anwendung, zusammen mit den geforderten Parametern. Die obigen Überlegungen führen dann zu den folgenden formalen Definitionen.

Definition 5 (Dienstbeschreibung). *Eine* Dienstbeschreibung $\langle networkaddress :: name \quad type_1, \ldots, type_n \rangle$ *ist eine Repräsentation eines über Netzwerke durchzuführenden Prozeduraufrufes, gegeben durch eine Netzwerkaddresse eines Servers, den Namen des anzufordernden Dienstes und den Typen der Parameter.*

Definition 6 (Dialogzielbeschreibung). *Eine* Dialogzielbeschreibung *ist ein Tupel* $\langle D, D', \mathbf{M}, \{D_1, \ldots, D_n\}, \{\mathbf{M}_1, \ldots \mathbf{M}_n\}, min, max \rangle$ *wobei die Minimale Information* D*, die Default Information* D' *und die Unterzielbeschreibungen* D_i *Merkmalsstrukturen, und* min *und* max *natürliche Zahlen sind so daß*

1. D *und* D' *kompatibel sind,*
2. $D_i \sqsubset D$*,*
3. $min < max$*,*

und $\mathbf{M}$ *und* $\mathbf{M}_i$ *Mengen von Dienstbeschreibungen sind, die dem Dialogziel respektive dem* i*-ten Unterziel zugeordnet sind.*

Die Defaultinformation muß kompatibel mit der minimalen Information sein, damit, falls Defaultinformation zum Dienstaufruf verwendet wird, eine konsistente Beschreibung der Intention des Benutzers erhalten wird. Die Unterziele müssen weniger Information für ihre Ausführung benötigen so daß die Ausführung der Unterziel-Dienste vor der Ausführung der Dialogziel-Dienste gewährleistet ist.

5.4.2 Das Aufgabenmodell

Wie oben schon erwähnt, ist das Aufgabenmodell nicht viel mehr als eine Menge von Dialogzielbeschreibungen.

Definition 7 (Aufgabenmodell). *Das* Aufgabenmodell *für eine gegebene Anwendung besteht aus einer Menge* $\{\mathbf{G}_1, \ldots, \mathbf{G}_n\}$ *von Dialogzielbeschreibungen* $\mathbf{G}_i$ *so daß die folgenden Bedingungen erfüllt sind:*

(i) Nichttriviale Zielbeschreibungen
$\bot \sqsubset D_i$ *für alle* i*.*

(ii) Unabhängige Zielbeschreibungen
Für die minimale Information D_i, D_j der Dialogzielbeschreibungen $\mathbf{G}_i, \mathbf{G}_j$, und $i \neq j$ gilt $D_i \not\sqsubseteq D_j$.

Nichttriviale Zielbeschreibungen werden gefordert, um sicherzustellen, daß Dialog tatsächlich stattfinden kann und nicht die semantischen Repräsentation des ersten Äußerung des Benutzers, unabhängig vom Inhalt, eine Dialogzielbeschreibung trivialerweise erfüllt. Unabhängige Zielbeschreibungen werden gefordert

Unabhängige Zielbeschreibungen werden gefordert, um sicherzustellen, daß alle Dialogziele erreicht werden können. Um zu sehen, warum, nehmen wir an, daß zwei Zielbeschreibungen $\mathbf{G}_1, \mathbf{G}_2$ mit minimaler Information D_1, D_2 existieren, sodaß $D_1 \sqsubseteq D_2$. Wie oben beschrieben, stehen D_1 und D_2 für die minimal Information, die im Diskurs vorhanden sein muß, um die mit $\mathbf{G}_1, \mathbf{G}_2$ assoziierten Dienste anzufordern. Ist die Information im Diskurs spezifisch genug, so daß sie von D_1 subsumiert wird, ist die Dialogzielbeschreibung $\mathbf{G}_1$ erreicht und die $\mathbf{G}_1$ zugeordneten Dienste werden angefordert. Sowie das Dialogsystem wieder die Kontrolle übernimmt, nimmt es richtigerweise an, daß ein neuer Dialog startet, da ja ein Dialogziel erreicht worden ist. Dies bedeutet, daß $\mathbf{G}_2$ niemals erreicht werden kann.

In der vorliegenden Implementierung werden die Bedingungen an das Aufgabenmodell vor Systemstart überprüft.

Als weiterer Kommentar sei vermerkt, daß das Aufgabenmodell zur Laufzeit ausgetauscht werden kann. Damit ist es zum Beispiel möglich, in einer ECommerce Anwendung verschiedene Aufgaben während der Auswahl von Gütern (wie zum Beispiel in den Einkaufskorb legen) und während des Bezahlens zu haben.

5.5 Zusammenfassung

Die drei in diesem Kapitel beschriebenen Interaktionsformen sowie die objektorientierten Erweiterungen der Merkmalsstrukturen können durch die Regel

> if *available information* is at-least-as-specific-as *some bound* and
> *some device dependent condition is met* then *start interaction.*

beschrieben werden.

Wenn man statt der Interaktionsformen diese Regel betrachtet, werden die Vorzüge des verfolgten Ansatzes sofort klar: das Verhalten des Dialogsystems kann ganz allein *über Subsumptions- und Kompatibilitätsrelationen der im Diskurs vorhanden Information mit den Komponenten des Dialogsystems* beschrieben werden. Da die Repräsentationen selbst, die typisierten Merkmalsstrukturen, wiederum von der Typenhierarchie (siehe Abschnitt 2.2) parametrisiert werden, kann die Beschreibung des Systemverhaltens sprach- und domänenunabhängig erfolgen.

Der zweite Teil dieser Arbeit beschäftigt sich damit, wie genau das Systemverhalten beschrieben wird.

Teil II

Ein Katalog von Interaktionsmustern

Kapitel 6

Informationelle Klassifikation der Dialogzustände

Im zweiten Teil dieser Arbeit wird beschrieben, wie eine allgemeine, d.h. domänen- und sprachunabhängige Dialoglogik die im ersten Teil beschriebenen Dienste anfordert, um ein Dialogsystem zu implementieren.

Damit ein Dialogsystem in scheinbar intelligenter Weise auf Eingaben reagieren kann, muß eine Repräsentation des Dialogzustandes vorliegen. In diesem Kapitel wird beschrieben, wie dieser Dialogzustand domänen- und sprachunabhängig repräsentiert werden kann. Als ein Beispiel betrachte man zwei verschiedene Dialogsysteme, zum Beispiel einen Stadtauskunftskiosk und ein Zugreservierungssystem. Das Stadtinformationssystem bietet die Möglichkeit an, Hotel- und Restaurantinformationen anzuzeigen, oder Fahrtanweisungen zu Zielorten zu geben. Das Zugreservierungssystem erlaubt es dem Benutzer, Zugauskünfte abzufragen oder Reservierungen vorzunehmen oder zu widerrufen. Offensichtlich sind die angebotenen Dienstleistungen, das verwendete Vokabular und die verfügbaren Datenbanken in beiden Anwendungen unterschiedlich. Vom Standpunkt der Dialogverarbeitung hingegen weisen beide Systeme genügend Ähnlichkeiten auf, so daß diese Ähnlichkeiten isoliert und generalisiert werden können.

Nehmen wir an, der Benutzer mache die folgenden Äußerungen zu jeweiligen System:

(i) Show me some middle class hotels downtown
(ii) I want to make a reservation for tomorrow

Nehmen wir weiter an, daß die unterstrichenen Teile vom Dialogsystem ignoriert werden, zum Beispiel durch Fehlerkennung oder fehlende Grammatikabdeckung. In beiden Fällen ist unklar, was die Intention des Benutzers ist: im ersten Fall könnte der Benutzer eine Hotelreservierung vornehmen wollen, oder den kürzesten Weg zu einem Restaurant angezeigt bekommen wollen. Im zweiten Fall könnte es sich um eine Reservierung oder deren Widerruf handeln. In beiden Fällen muß das Dialogsystem weitere Information akquirieren, bevor es die Dienste der Anwendung anfordern kann. Es ist aus der Sichtweise von allgemeinen Dialogstrategien von daher sinnvoll, diese Gemeinsamkeit in

einem abstrakten Dialogzustand zu repräsentieren. Mit anderen Worten, die interne Zustandsrepräsentation in beiden System *ist die gleiche in Bezug auf die nächste Aktion* des Dialogsystems.

Um das Beispiel fortzusetzen, nehmen wir an, daß das System in beiden Fällen eine Klärungsfrage stellt. Die Antwort des Benutzer ist

(i) Display on the map
(ii) Book the reservation

Nachdem in beiden Fällen die semantischen Repräsentationen erzeugt und in den Diskurs inkorporiert wurden, sind beide Systeme wieder in demselben Dialogzustand *bezüglich der nächsten Aktion*, die vom Dialogsystem unternommen werden muß. Jetzt verfügt das Dialogsystem über genügend Wissen, um die vom Benutzer gewünschte Aktion zu bestimmen, hat jedoch nicht genug Information, um die Aktion auszuführen (Hotelname und Abfahrtsort, respektive, fehlen).

Es ist jedoch nicht wirklich nötig, die explizit modellierten Zustände zu durchwandern. Für die Dialogverarbeitung ist es vielmehr wichtig zu bestimmen, ob eine gegebene Eigenschaft im gegenwärtigen Zustand wahr ist oder nicht. Im obigen Beispiel sind die beiden betrachteten Eigenschaften, ob das Dialogziel eindeutig bestimmt ist und ob zur Ausführung des Dialogzieles noch Information fehlt. Es ist offensichtlich, daß die Dialogstrategie in Abhängigkeit dieser Eigenschaften formuliert werden kann, und eine explizite Modellierung des Dialogzustandes nicht benötigt wird. Nehmen wir weiter an, daß das oben beschriebene Dialogsystem so erweitert werden soll, daß bei Konfidenzannotationen unter 80 Prozent eine Bestätigungsfrage gestellt werden soll. Nehmen wir an, daß n Felder erfragt werden müssen. Dann gibt es im endlichen Automaten im schlimmsten Fall 2^n Zustände (die Information für jedes Feld kann bekannt oder unbekannt sein). Wird jetzt jedoch die Information über die Konfidenzannotation hinzugefügt, steigt die Anzahl der Zustände auf 3^n (der Wert für jedes Feld kann unbekannt sein, oder bekannt mit Konfidenz kleiner gleich 80 Prozent oder bekannt mit Konfidenz über 80 Prozent).

Für die Entscheidungsfindung in der Dialogstrategie ist es aber nicht erforderlich, all diese Information in Betracht zu ziehen. Es ist vielmehr erforderlich zu wissen, ob es ein Feld mit Konfidenz kleiner gleich 80 Prozent gibt, um eine entsprechende Bestätigungsfrage zu generieren. Für welches Feld die Frage generiert werden muß, kann dann lokal vom Generierungsalgorithmus bestimmt werden.

6.1 Einleitung

In den Kapiteln 2 und 3 wurden verschiedene Repräsentationsformen für Information im Diskurs beschrieben. In Kapitel 4 wurde beschrieben, wie natürlichsprachlicher Text in Repräsentationen und zurück konvertiert werden kann. In Kapitel 5 wurde die Konvertierung zwischen Datenbanken und Repräsentationen beschrieben. Dieses Kapitel charakterisiert den Dialogzustand *in Abhängigkeit* der im ersten Teil der Arbeit beschriebenen Prozesse. Da diese Prozesse

domänen- und sprachunabhängig definiert worden sind, ist es möglich, auch die Charakterisierung domänen- und sprachunabhängig zu halten.

Das Ziel der Charakterisierung des Dialogzustandes ist es, ein Maß des Fortschritts des Dialogs zu erhalten. Mit anderen Worten, die Charakterisierung des Dialogzustandes beantwortet die Frage: *"Wie dicht ist der Dialog an der Ausführung eines Dialogzieles?"*. Natürlich ist die Charakterisierung des Dialogzustandes nur Mittel zu dem Zweck, die nächste Aktion des Dialogsystems zu beschreiben. Wie dies genau geschieht, ist Gegenstand des folgenden Kapitels. Es ist jedoch notwendig, diese Motivation nicht aus den Augen zu verlieren, da die Charakterisierung des Dialogzustandes fein genug sein muß, um alle gewünschten Aktionen ausdrücken zu können.

Die Charakterisierung des Dialogzustandes geschieht durch eine Menge von Zustandsvariablen. Jede Zustandsvariable mißt einen Aspekt des Dialogs. Der Dialogzustand ist dann gegeben durch die Wertebelegung der Zustandsvariablen.

6.2 Konsistenz und Qualität der Eingabe: Zustandsvariable TurnQuality

Die Eingabe zu dem Dialogsystem wird von einem Spracherkenner bereitgestellt. Beim heutigen Stand der Technik treten Spracherkennungsfehler noch häufig auf; dies kann zum Beispiel durch Dialekt, *Out of vocabulary words* oder andere Ursachen begründet sein. Offensichtlich führen Spracherkennungsfehler zu verringerter Effektivität des Dialogsystems. Es ist von daher wünschenswert, das Auftreten von Spracherkennungsfehlern zu erkennen und im Dialog entsprechend zu reagieren.

6.2.1 Definition der Zustandsvariable

Die Qualität der Eingabe wird in der Zustandsvariable TurnQuality erfaßt. Es ist Aufgabe der Variable zu beschreiben, ob die semantische Repräsentation der letzten Eingabe ignoriert werden soll oder nicht. Desweiteren ist im letzteren Fall anzugeben, ob Teile der Repräsentation verwendet werden sollen. Sollen Teile der Repräsentation ignoriert werden, kann es hilfreich sein zu beschreiben, ob die geringe Qualität dieser Teile durch geringe Konfidenzbewertungen oder inkonsistente Repräsentationen begründet sind. Damit ergibt sich die folgende Definition der Zustandsvariable.

Definition 1 (Zustandsvariable TurnQuality). *Die Zustandsvariable* TurnQuality *beschreibt die Qualität der semantischen Repräsentation des gegenwärtigen Turns. Die Werte, die die Variable annehmen kann, sind (i)* good_quality*, (ii)* low_confidence*, (iii)* partially_inconsistent_representation *und (iv)* poor_quality*, jeweils mit den offensichtlichen Bedeutungen.*

Bevor auf die Bestimmung des Variablenwertes eingegangen wird, soll noch einmal erwähnt werden, daß eine Variablenbelegung keinerlei Aktion vorschreibt. Im Gegenteil, es ist Aufgabe der Zustandsvariablen den gegenwärtigen

Zustand fein genug zu klassifizieren, so daß im nächsten Schritt eine Dialogstrategie (die bezüglich der Zustandsvariablen formuliert ist) die nötigen Aktionen auslösen kann. Damit kann eine einfache Anpassung an verschiedene System-Anforderungen vorgenommen werden. In manchen Anwendungen, wie zum Beispiel der Eingabe von Kreditkarten-Informationen, ist es nötig, im Falle von inkonsistenter Eingabe die gesamte Eingabe wiederholen zu lassen. In anderen Anwendungen können die inkonsistenten Teile der Repräsentationen ignoriert und die ignorierte Information durch Klärungsfragen bestimmt werden. Dieses Beispiel zeigt, daß Zustandsvariablen die Repräsentation des Dialogzustandes von der Repräsentation der Aktionen des Systems trennt.

6.2.2 Bestimmen der Variablenbelegung

In dieser Arbeit werden im wesentlichen drei Merkmale verwandt, um das Auftreten von Spracherkennungsfehlern zu erkennen. Zum einen kann die Hypothese mit Konfidenzmaßen annotiert werden. Ist das Vertrauen des Erkenners unterhalb eines gewissen Schwellwertes, wird ein Teil der Hypothese – oder gegebenenfalls die gesamte Hypothese ignoriert.

Darüber hinaus gibt der Parser Auskunft darüber, wieviel Worte der Eingabe durch Grammatikregeln abgedeckt werden konnten. Damit kann erfaßt werden, ob Teile der Eingabe zwangsweise ignoriert werden müssen, da sie nicht in semantische Repräsentationen umgewandelt werden können.

Schließlich kann die vom Parser erzeugte semantische Repräsentation für die Detektion von Fehlerkennungen verwandt werden. Wie in Abschnitt 4.3.1 erläutert, garantiert das Format der Grammatikregeln, daß eine grammatikalische[1] Eingabe eine konsistente und wohlgetypte Merkmalsstruktur erzeugt. Dies bedeutet aber, daß eine inkonsistente oder nicht wohltypisierbare Merkmalsstruktur durch einen ungrammatikalischen Satz erzeugt worden sein muß. In diesem Fall können Teile der erstellten Repräsentation zurückgewiesen werden.

6.3 Gesamtqualität des Dialogs

Um ein vollständiges Bild des fortschreitenden Dialogs zu erhalten, ist es notwendig, nicht nur die Qualität der aktuellen Äußerung zu untersuchen. Darüber hinaus ist es auch notwendig, die Gesamtqualität des gesamten Dialogs zu berücksichtigen. Wenn beispielsweise die Qualität der Spracherkennung zu schlecht wird, steigt die Anzahl der Klärungs- und Bestätigungsfragen, und damit auch die Frustation des Benutzers. Die Beobachtung der Gesamtqualität erlaubt es dann in solchen Fällen beispielsweise, einen Anruf zu einem menschlichen Operator zu transferieren.

[1] *Grammatikalisch* ist in diesem Zusammenhang relativ zu der verwendeten Grammatik zu verstehen. So ist beispielsweise eine Eingabe, die von der Grammatik vollständig analysiert wird, grammatikalisch bezüglich der verwendeten Grammatik, auch wenn der Eingabesatz kein korrekter Satz des Deutschen ist.

6.3.1 Definition der Zustandsvariable

Die Gesamtqualität des fortschreitenden Dialogs wird durch eine Zustandsvariable OverallQuality erfaßt. Das Intervall der möglichen Werte, die die Variable annehmen kann, reicht von *poor* zu *perfect*. Die minimale Anzahl der Interaktionen zwischen Benutzer und Dialogsystem bestimmt dann die Anzahl der Zwischenwerte. Wenn das Dialogsystem beispielsweise nach drei fehlgeschlagenen Interaktionsversuchen den Dialog abbrechen soll, muß es zwei weitere Werte $intermediate_1$ und $intermediate_2$ geben. Feinere Abstufungen sind möglich, falls erforderlich. Damit ergibt sich folgende Definition der Zustandsvariable.

Definition 2 (Zustandsvariable *OverallQuality*). *Die Zustandsvariable* OverallQuality *beschreibt die Gesamtqualität des fortschreitenden Dialogs. Die Wertebelegung der Variable umfaßt die Werte* poor *und* perfect *sowie weitere Zwischenwerte* $intermediate_1$ *und* $intermediate_2$.

6.3.2 Bestimmen der Variablenbelegung

Die initiale Variablenbelegung bei Beginn des Dialogs ist *perfect*. Der Wert der Variable wird in Abhängigkeit des Wertes der Variable TurnQuality angepaßt. Ist der Wert von TurnQuality *good_quality*, wird der Wert der Variable OverallQuality unverändert gelassen. Anderenfalls wird der Wert der Variable OverallQuality auf den Vorgänger des gegenwärtigen Wertes gesetzt.

6.4 Der Sprechakt der aktuellen Äußerung

Sprechakterkennung ist seit langem Gegenstand der Forschung [Traum and Hinkelman, 1992,Traum, 1999]. Sprechakterkennung ist in Bezug auf natürlichsprachliche Dialogsysteme deswegen wichtig, weil der Sprechakt einer Äußerung – zumindest teilweise – bestimmt, wie die semantische Repräsentation der Äußerung in den Diskurs eingefügt werden soll und welche Aktionen das System in Antwort auf die gegenwärtige Äußerung ausführen soll. Dieser Punkt wird durch folgendes Beispiel erläutert. Wenn das Stadtinformationskiosk etwa die Klärungsfrage stellt *"Möchten Sie im Hotel Kaiser, im Stadthotel oder im Holiday Inn ein Zimmer reservieren?"*, wird versucht, die folgende Äußerung des Benutzers als Antwort auf die Klärungsfrage zu interpretieren. Antwortet der Benutzer hingegen mit einer Gegenfrage, so zum Beispiel *"Welches Hotel ist am billigsten?"* so kann eine Sprechakterkennung die Fehlinterpretation der Äußerung als Antwort verhindern. Damit dient die Sprechakterkennung auch als Indikator für die Diskursstruktur [Grosz and Sidner, 1986], da, wie in diesem Beispiel, ein Unterdialog eröffnet werden kann.

6.4.1 Definition der Zustandsvariable

Der Sprechakt der gegenwärtigen Äußerung des Benutzers wird in der Zustandsvariable SpeechAct vermerkt. Die Wertebelegung der Variable, mögliche Sprechakte also, wird dann durch eine Sprechakt-Taxonomie gegeben. Das

Erstellen von Sprechakt-Taxonomien ist intensiv erforscht worden. Allerdings war das Ziel dieser Forschung zumeist, einen Sprechakt-Katalog zu erstellen, der die Sprechakte im Dialog von einem *linguistischen* Standpunkt aus optimal beschreibt. Die Motivation bei der hier zu verwendenden Sprechakt-Taxonomie ist jedoch zu beschreiben, welche Aktion das Dialogsystem am besten ausführen soll. Aus diesem Grunde wurde eine Sprechakt-Taxonomie eigens zu diesem Zweck entwickelt. Abbildung 6.1 zeigt die verwendete Sprechakt-Taxonomie.

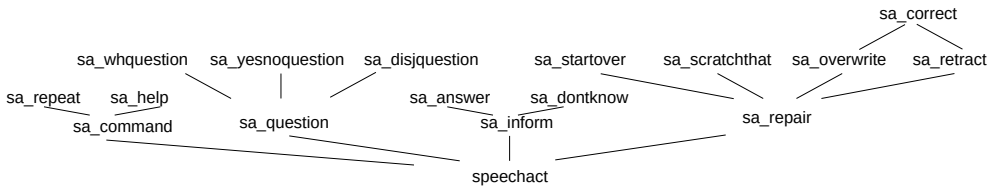


Abbildung 6.1. Die für die abstrakte Klassifizierung des Dialogzustandes verwendete Sprechakttaxonomie.

Damit ergibt sich die Definition der Zustandsvariable wie folgt.

Definition 3 (Zustandsvariable *SpeechAct*). *Die Zustandsvariable* SpeechAct *beschreibt den Sprechakt der aktuellen Äußerung des Benutzers. Die Variable kann einen der Sprechakte*

Die verwendeten Sprechakte zusammen mit ihrer Bedeutung sind in Tabelle 6.1 aufgelistet.

6.4.2 Bestimmen der Variablenbelegung

Der Wert der SpeechAct Variable wird aus einer Reihe von Faktoren bestimmt. Teilweise wird der Sprechakt aus der vom Parser erzeugten semantischen Repräsentation erzeugt. Ist dies nicht möglich, wird überprüft, die aktuelle Aussage eine Erwiderung auf eine Frage ist. Ist das der Fall und ist die semantische Repräsentation des Sprechaktes kompatibel mit der Repräsentation, wird angenommen, daß der Sprechakt die Antwort auf die Frage stellt. Sind hingegen die Repräsentationen nicht kompatibel, wird angenommen, daß der Benutzer eine Gegenfrage gestellt hat.

6.5 Bestimmen des Datenbankzugriffs

Das Dialogsystem hat die Möglichkeit, Referenten für definite Beschreibungen durch Datenbankanfragen zu bestimmen. Wie in Abschnitt 5.3 beschrieben, wird der Zugriff auf die Datenbanken durch die Datenbankwächter gesteuert. Ist ein Datenbankzugriff für eine gegebene definite Beschreibung nötig, aber nicht genügend Information vorhanden, um ihn auszuführen, müssen entsprechende Information erfragt werden. Um die Generierung der entsprechenden Fragen zu ermöglichen, muß der abstrakte Dialogzustand die Relation der im Diskurs vorhandenen Information bezüglich der vorhandenen Datenbanken repräsentieren. Damit ergibt sich die Notwendigkeit der Zustandsvariable Reference.

Gruppe	Sprechakt	Bedeutung
sa_command	*sa_repeat*	Aufforderung an das System, die letzte Äußerung zu wiederholen
	sa_help	Hilfeanforderung
sa_question	*sa_whquestion*	Ersetzungsfrage
	sa_yesnoquestion	Ja/Nein Frage
	sa_disjquestion	Disjunktivfrage
sa_inform	*sa_answer*	Antwort auf eine Frage
	sa_dontknow	Verweigerung der Antwort
sa_repair	*sa_startover*	versetzt das Dialogsystem in den Ausgangszustand
	sa_scratchthat	macht die Effekte der letzten Äußerung rückgängig
	sa_overwrite	ersetzt einen implizit bestimmten Teil des Diskurses durch explizit gegebene Information *"No I meant x"*
	sa_retract	entfernt einen explizit bestimmten Teil des Diskurses *"No not x"*
	sa_correct	eine Kombination der Akte *sa_overwrite* und *sa_correct* *"No not x I meant y"*

Tabelle 6.1. Verwendete Sprechakte, deren Bedeutung und ihre Erkennung.

6.5.1 Definition der Zustandsvariable

Die Zustandsvariable REFERENCE drückt aus, ob für die Beschreibungen unter Diskussion noch Datenbankzugriffe ausgeführt werden müssen. Damit kann die Variable die folgenden Werte annehmen. Der Wert *deselected* zeigt an, daß es keine Datenbanken gibt, die Beschreibungen im Diskurs auflösen können. Dies ist unter anderem der Fall, wenn es keine Repräsentation einer Beschreibungen im Diskurs gibt, für die ein Datenbankzugriff durchgeführt werden kann. Dies ist beispielsweise bei kurzen Kommandos wie *"Wiederholen"* oder *"Hilfe!"* der Fall. Beide Kommandos enthalten keine definiten Beschreibungen, somit ist kein Datenbankzugriff nötig. Der Wert *selected* wird angenommen, wenn es zwar aufzulösende Beschreibungen im Diskurs gibt, aber nicht genügend Information vorhanden ist, um den Datenbankzugriff durchzuführen. Der Wert *finalized* wird angenommen, nachdem der Datenbankzugriff durchgeführt worden ist. Das Übergangsdiagramm der Zustandsvariablen bei monoton wachsender Spezifizität ist in Abbildung 6.2 gegeben.

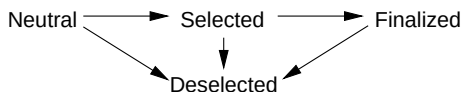


Abbildung 6.2. Erlaubte Zustandsübergänge der Zustandsvariable REFERENCE.

6.5.2 Bestimmen der Variablenbelegung

Die Variablenbelegung wird mithilfe der Subsumptionsrelation über Merkmalsstrukturen bestimmt. Hierzu werden die oben beschriebenen Fälle abgeprüft und der Wert der Variablen entsprechend gesetzt.

6.6 Auflösung referierender Ausdrücke

Im vorigen Abschnitt wurde beschrieben, wie in Gegenwart von Repräsentation referierender Ausdrücke im Diskurs die Datenbank für deren Auflösung bestimmt wird. Nachdem der Datenbankzugriff durchgeführt worden ist, muß erfaßt werden, ob die Referenz der Beschreibungen eindeutig ist oder nicht. Beispielsweise kann die Beschreibung *"das Hotel"* auf mehrere Objekte referieren,aber zur Angabe des kürzesten Weges benötigt das System eine eindeutige Objektbeschreibung. Dies motiviert die Einführung einer weiteren Zustandsvariablen ReferringExpressions.

6.6.1 Definition der Zustandsvariable

Die Eigenschaften referierender Ausdrücke sind wie folgt klassifiziert. Der Wert *UniqueReference* beschreibt den Zustand, in dem es nur ein Objekt gibt, welches die Constraints des Benutzers erfüllt. Der Wert *AmbiguousReference* beschreibt den Zustand, in dem mehrere Objekte der Datenbank den Anforderungen des Anwenders entsprechen. Der Wert *ImperfectReference* beschreibt den Zustand, in dem kein Objekt genau den Anforderungen des Benutzers entspricht, es aber "ähnliche" Objekte gibt. Der Wert *NoReference* beschreibt den Zustand, in dem kein Objekt den Anforderungen des Benutzers entspricht, und es auch keine "ähnlichen" Objekte gibt.

Bei monoton wachsender Spezifizität der Information im Diskurs ergeben sich die Zustandsübergänge der Variable zu den in Abbildung 6.3 gezeigten.

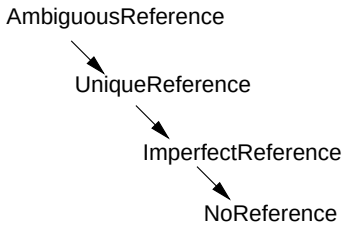


Abbildung 6.3. Die Zustandsübergänge der Variablenwerte für die Variable ReferringExpressions

Die Ordnung zeigt den Grad der vorhandenen Information im Diskurs an: je mehr Information im Diskurs vorhanden ist, desto mehr bewegt sich der Wert der Zustandsvariable nach rechts. Die Definition der Zustandsveriable ist dann wie folgt.

Definition 4 (Zustandsvariable REFERENCE**).** *Die Zustandsvariable*Reference *beschreibt die Relationen der referierenden Ausdrücke im Diskurs zu den Objekten, auf die referiert wird. Hierbei wird der Wert der Variable auf das Minimum bezüglich der in Abbildung 6.3 gezeigten Ordnung gesetzt, falls mehr als ein referierender Ausdruck vorhanden ist.*

6.6.2 Bestimmen des Wertes der Zustandsvariablen

Der Wert der Zustandsvariable wird wie folgt bestimmt. Zunächst wird für alle referierende Ausdrücke, für die ein Datenbankzugriff möglich ist, dieser durchgeführt. Dann wird für jeden referierenden Ausdruck die Relation zwischen referierendem Ausdruck und den Objekten, auf die referiert wird, bestimmt. Schließlich wird die Variable auf das Minimum der so bestimmten Werte gesetzt.

6.7 Bestimmen der Intention des Benutzers

Wie in Abschnitt 5.4 beschrieben, ist eine Dialogzielbeschreibung definiert durch die minimale Menge an Information, die notwendig ist, um die dem Ziel zugeordneten Dienste auszuführen.

Während des Dialogs des Benutzer mit dem System ist es die Aufgabe des Dialogsystems:

1. zu bestimmen, ob der Benutzer einen der angebotenen Dienste in Anspruch nehmen möchte, und falls dem so ist,
2. interaktiv im Dialog mit dem Benutzer genügend Information zu akquirieren um den intendierten Dienst und alle nötigen Parameter eindeutig zu bestimmen, und
3. schließlich den eindeutig bestimmten Dienst von dem entsprechenden Subsystem anzufordern.

Um den Fortschritt des Dialogs bezüglich der akquirierten Information zu beschreiben, ist jede Dialogzielbeschreibung mit einer Zustandsvariable versehen. Die Zustandsvariable kann einen der folgenden Werte annehmen.

Definition 5 (Zustände der Dialogzielbeschreibungen). *Zu jedem Zeitpunkt befindet sich eine Dialogzielbeschreibung in einem der folgenden Zustände.*

1. **Neutral**
   Im Diskurs ist keinerlei Information repräsentiert.
2. **Selected**
   Die im Diskurs vorhandene Information is kompatibel mit der Dialogzielbeschreibung, und es gibt wenigstens eine weitere Dialogzielbeschreibung, die mit der Information im Diskurs kompatibel ist.
3. **Deselected**
   Die Dialogzielbeschreibung ist inkompatibel mit der Information im Diskurs.

4. **Determined**
   Die Information im Diskurs ist kompatibel mit der Dialogzielbeschreibung, und es gibt keine weitere Dialogzielbeschreibung, mit der die Information im Diskurs kompatibel ist, das heißt, es gibt keinen andere Dialogzielbeschreibung in den Zuständen Selected, Determined *oder* Finalized.
5. **Finalized**
   Die Information im Diskurs wird von der Dialogzielbeschreibung subsumiert.

6.7.1 Definition der Zustandsvariable

Für die Dialogverarbeitung ist es wichtig, den Zustand derjenigen Dialogzielbeschreibung zu kennen, die dem Endzustand am nächsten gekommen ist.

Definition 6 (Zustandsvariable *Intention*). *Die Zustandsvariable* Intention *enthält den Zustand derjenigen Dialogzielbeschreibung, die dem Zustand* Abgeschlossen *am nächsten gekommen ist, oder* inkonsistent*, falls sich alle Dialogzielbeschreibungen im Zustand* Deselektiert *befinden. Die erlaubten Werte der Zustandsvariable sind demzufolge* NEUTRAL, SELEKTIERT, EINDEUTIG BESTIMMT, ABGESCHLOSSEN *und* INKONSISTENT.

6.7.2 Bestimmen des Wertes der Zustandsvariable

Der Wert der Variable ist der Zustandswert derjenigen Dialogzielbeschreibung, die die Intention des Benutzers am besten beschreibt. Mit anderen Worten, der Wert wird wie folgt bestimmt.

1. wenn der Zustand aller Dialogzielbeschreibungen *neutral* ist, ist der Wert der Variable INTENTION ebenfalls *neutral*;
2. wenn es eine Dialogzielbeschreibung im Zustand *selected* gibt, dann ist der Wert der Variable INTENTION ebenfalls *selected* (dies impliziert bei Definition 5 daß es keine Dialogzielbeschreibung gibt, die sich im Zustand *determined* oder *finalized* befindet);
3. gibt es eine Dialogzielbeschreibung im Zustand *determined*, dann wird INTENTION auf den Wert *determined* gesetzt (man bemerke, daß dies bei Definition 5 impliziert, daß weder eine Dialogzielbeschreibung im Zustand SELECTED noch im Zustand FINALIZED ist);
4. gibt es eine Dialogzielbeschreibung im Zustand FINALIZED, dann wird INTENTION auf den Wert FINALIZED gesetzt (man bemerke, daß dies bei Definition 5 impliziert, daß weder eine Dialogzielbeschreibung im Zustand *selected* noch im Zustand DETERMINED ist);
5. anderenfalls sind alle Dialogzielbeschreibungen im Zustand *deselected*, was impliziert, daß keine Dialogzielbeschreibung die Intention des Benutzers korrekt wiedergibt. In diesem Falle wird die Variable INTENTION auf den Wert *inconsistent* gesetzt.

Wenn wir monoton wachsende Spezifizität der Information im Diskurs annehmen, sind die erlaubten Zustandsübergänge die in Abbildung 6.4 gezeigten. Die Anwendung wird von stattfindenden Zustandsübergangen benachrichtigt,

so daß das Einbinden von Information im Diskurs sowohl durch verbale Bestätigung als auch durch eine Aktion der Anwendung (beispielsweise visuell) quittiert werden kann.

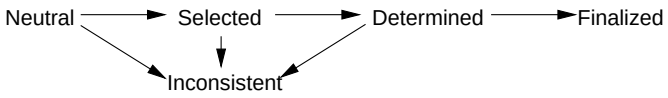


Abbildung 6.4. Erlaubte Zustandsübergänge der Variable INTENTION bei monoton wachsender Spezifizität.

6.8 Informationelle Klassifikation der Dialogzustände

In der Tabelle 6.2 sind die in den vorhergehenden Abschnitten beschriebenen Variablen, deren möglich Belegungen und deren Bedeutung zusammengefaßt.

Zustandsvariable	**Werte**	**Bedeutung**
TURNQUALITY	*good,inconsistent, lowconfidence,poor*	Qualität der Repräsentation der aktuellen Äußerung
OverallQuality	*good, intermediate*$_1$ *intermediate*$_2$*,poor*	Qualität des Gesamt-Dialogs
SpeechAct	*sa_answer, sa_yesnoquestion,...*	Sprechakt der aktuellen Äußerung
Reference	*neutral, selected determined, finalized*	Zustand des Prozesses des Auflösens von Referenzen
ReferringExpressions	*NoReference, ImperfectReference UniqueReference,AmbiguousReference*	Zustand der referierenden Ausdrücke
Intention	*neutral, selected determined, finalized, inconsistent*	Grad zu dem die Intention des Anwenders bestimmt werden konnte

Tabelle 6.2. Überblick über die Zustandsvariablen, ihre Werte und ihre Bedeutung

Der abstrakte Dialogzustand eines Dialoges ist dann gegeben durch die Variablenbelegung der Zustandsvariablen.

Definition 7 (Abstrakter Dialogzustand). *Der* abstrakte Dialogzustand *ist gegeben durch die Variablenbelegung* $\langle v_1, \ldots, v_6 \rangle$ *der Variablen* TURNQUALITY*,* OVERALLQUALITY*,* SPEECHACT*,* REFERRENCE*,* REFERRINGEXPRESSIONS *und* INTENTION*.*

Im nächsten Kapitel wird beschrieben, wie anhand des abstrakten Dialogzustandes Aktionen des Systems ausgelöst werden können.

6.9 Ein Beispiel

Im folgenden wird ein Beispiel gegeben, welches die Bestimmung des abstrakten Dialogzustandes erläutern soll. Wir nehmen als System ein Autonavigationssystem an. Es gibt drei Dialogzielbeschreibungen, nämlich GiveDirections, DisplayObjects und WhereAmI, welche es ermöglichen, den gegenwärtigen Aufenthaltsort zu erfragen, Objekte auf einer Karte anzuzeigen oder Wegbeschreibungen zu einem Zielort zu generieren. Es gibt Datenbanken für Museen, Restaurants und Bars. Die Restaurant-Datenbank hat die in Abbildung 5.4 gezeigten Datenbankwächter. Wir lassen an dieser Stelle offen, wie das System seine nächste Aktion entscheidet, da dies erst in den folgenden Kapitel beschrieben wird. An dieser Stelle werden lediglich die Zustandsvariablen angezeigt. Die gezeigte semantische Repräsentation ist die Repräsentation der Äußerungen des Anwenders, kumuliert über die Dauer des Dialogs.

Anwender: I want to have Burritos for dinner
Repräsentation: $[\bot]$
Dialogzustand: $\langle poor, intermediate_2, speechact, selected, none, selected\rangle$
System: I am sorry I did not understand everything you said. Please say "give directions", "display" or "where am I".

Der Spracherkenner hat die Eingabe nicht richtig erkennen können. Demzufolge hat der robuste semantische Parser keine Ableitung finden können und eine leere semantische Repräsentation erzeugt. Die Information im Diskurs ist nicht ausreichend, um einen Datenbankzugriff durchführen zu können. Alle Dialogzielbeschreibungen sind mit der Information im Diskurs kompatibel.

Anwender: Give directions.
Repräsentation: $[act\_givedirections]$
Dialogzustand: $\langle good, intermediate_2, sa\_answer, selected, none, determined\rangle$
System: Would you like to go to a bar, a restaurant or a museum?

Die Erkennung ist gut, ebenso konnte der Sprechakt gut erkannt werden. Es ist nach wie vor nicht genügend Information vorhanden, um einen Datenbankzugriff durchführen zu können, deswegen bleiben die Werte der Variablen Reference und ReferringExpressions unverändert. Die semantische Repräsentation schließt zwei der drei Dialogzielbeschreibungen als mögliche Repräsentation der Intention des Anwenders aus; allerdings ist noch nicht alle notwendige Information für die Dienstanforderung vorhanden, da der Zielort fehlt. Deswegen ist der Wert der Variablen Intention *determined*.

Anwender: A restaurant.
Repräsentation:
$$\begin{bmatrix} act\_givedirections \\ \text{PATH}\begin{bmatrix} obj\_path \\ \text{DST}\ obj\_restaurant \end{bmatrix} \\ \\ \end{bmatrix}$$
Dialogzustand: $\langle good, intermediate_2, sa\_answer, determined, none, determined\rangle$
System: What kind of food would you like? Please tell me the nationality.

Die Information im Diskurs ist ausreichend genug, um die Datenbanken für Bars und Museen auszuschließen, aber nicht spezifisch genug, um die Datenbankwächter für die Restaurant-Datenbank zu erfüllen. Deshalb nimmt die Variable Reference den Wert *Determined* an. Alle anderen Werte bleiben unverändert.

Anwender: Mexican.

Repräsentation:

$$\begin{bmatrix} act\_givedirections \\ \text{PATH} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant \\ \text{NATIONALITY } prp\_mexican \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Dialogzustand: $\langle good, intermediate_2, sa\_request, finalized, none, determined \rangle$
System: *System führt Datenbankanfrage aus.*
Dialogzustand: $\langle good, intermediate_2, sa\_answer, finalized, ambiguous, determined \rangle$
System: Would you like to go to a Mexican restaurant in Oakland, Shadyside or North Hills?

Die Information ist ausreichend genug, um den Datenbankzugriff durchzuführen, folglich erhält die Variable REFERENCE den Wert *finalized* nach dem Datenbankzugriff. Der Datenbankzugriff erzeugt eine Ergebnismenge mit fünf Objekten, deswegen nimmt die Variable REFERRINGEXPRESSIONS den Wert *ambiguous* an. Alle anderen Werte bleiben unverändert.

Anwender: Oakland.

Repräsentation:

$$\begin{bmatrix} act\_givedirections \\ \text{PATH} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant \\ \text{NATIONALITY } prp\_mexican \\ \text{ADDRESS} \begin{bmatrix} address \\ \text{NEIGHBORHOOD } \text{“}Oakland\text{“} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Dialogzustand: $\langle good, intermediate_2, sa\_answer, finalized, unique, finalized \rangle$
System: I am giving you directions to “Mad Mex“ in Oakland.

Nach Disambiguierung der Objektbeschreibungen der Restaurants verbleibt nur ein passendes Objekt, deswegen erhält die Variable REFERRINGEXPRESSEIONS den Wert *unique*. Außerdem sind die Constraints der Dialogzielbeschreibung nun erfüllt, folglich nimmt die Variable INTENTION den Wert *finalized* an.

Dieses Beispiel zeigt Schritt für Schritt, wie zusätzliche Information in den Diskurs integriert wird und die Zustandsvariablen angepaßt werden. Im folgenden Beispiel soll der datengetriebene Charakter der Dialogverarbeitung herausgestellt werden. Hier können Datenbankdienste früh im Dialog angefordert werden. Dadurch kann die Länge des Dialoges verkürzt werden.

Anwender: I need directions to a Mexican restaurant.

Repräsentation:

$$\begin{bmatrix} act\_givedirections \\ \text{PATH} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant \\ \text{NATIONALITY } prp\_mexican \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Dialogzustand: $\langle good, good, sa\_request, finalized, none, determined \rangle$
System: *System führt Datenbankanfrage aus.*
Dialogzustand: $\langle good, good, sa\_request, finalized, ambiguous, determined \rangle$
System: Would you like to go to a Mexican restaurant in Oakland, Shadyside or North Hills?

Nach diesem Dialog kann direkt zum vorletzten Schritt des vorigen Dialogs gesprungen werden.

6.10 Diskussion

Die gewählten Zustandsvariablen reflektieren wichtige Aspekte des fortschreitenden Dialogs. Bei der Wahl der Dialogzustände wurde darauf geachtet, daß sie sprach- und domänenunabhängige Aspekte des Dialogs beschreiben. Wie in den nächsten Kapiteln beschrieben, kann die Dialogstrategie dann angegeben werden durch Zustandsübergänge zwischen den abstrakten Dialogzuständen.

Die Abbildungen 6.5 und 6.6 illustrieren den Unterschied zwischen auf endlichen Automaten basierenden konventionellen Dialogzustandsdiagrammen und abstrakten Dialogzuständen. Beide Diagramme zeigen Zustandsübergänge, die folgendes Verhalten implementieren. Wenn die vom Benutzer bereitgestellte Information einen geringen Konfidenzwert hat, wird die letzte Frage wiederholt. Passiert dies zweimal, wird der Dialog abgebrochen. Wenn Information fehlt, wird eine Klärungsfrage gestellt. Wenn alle beide Slots gefüllt sind, ist das Dialogziel erreicht.

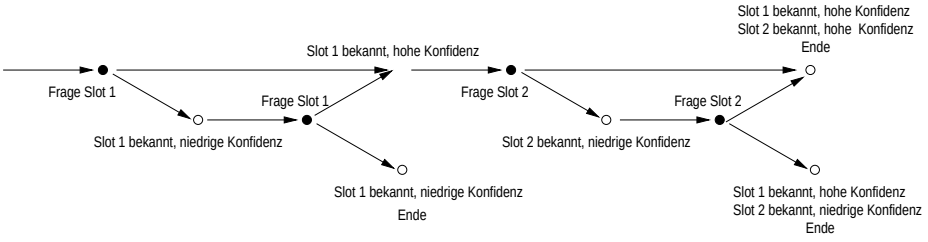


Abbildung 6.5. Zustandsübergänge in endlichen Automaten.

In Abbildung 6.5 gibt es konkrete Zuordnung der zu stellenden Fragen; dies ist in Abbilding 6.6 nicht der Fall. Die Zuordnung der vom Dialogsystem auszuführenden Aktion geschieht durch die Implementierung der Interaktionsmuster und wird in den folgenden Kapiteln beschrieben. Diese wiederum nehmen Zugriff auf die *Basis-Dialogdienste*, um die Oberflächenform der zu stellenden Fragen zu bestimmen.

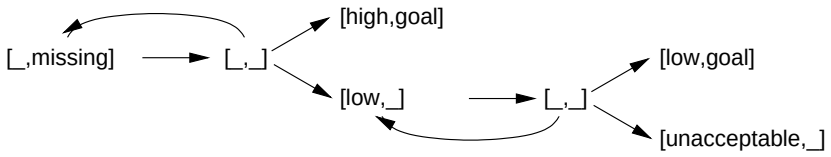


Abbildung 6.6. Abstrakte Dialogzustände und Zustandsübergänge.

Desweiteren fällt auf, daß schon im Falle von nur zwei Zustandsvariablen die Zahl der abstrakten Dialogzustände geringer ist als die der konkreten Zustände im konventionellen Übergangsdiagramm. Dieser Vorteil wird umso deutlicher, je mehr Zustandsvariablen zur Charakterisierung des Dialogzustandes verwendet werden.

Die einfachere Zustandsrepräsentation geht jedoch zuungunsten der Information: Im Zustandsübergangsdiagramm 6.5 ist zu jedem Zeitpunkt ablesbar, welche Information (Slot 1, Slot 2 oder beide) erfragt worden ist, im Übergangsdiagramm 6.6 ist dies nicht der Fall. Aus diesem Grunde gibt es in der hier vorgeschlagenen Architektur eine vom Dialogzustand unabhängige Repräsentation des Diskurses (siehe Kapitel 5).

Bei der Implementierung der Dialogstrategie hat sich dann gezeigt, daß alle hier beschriebenen Variablen notwendig sind, um das Verhalten des Systems beschreiben zu können. Das Auslassen von einer der beiden Variablen TurnQuality oder OverallQuality beispielsweise würde es nicht erlauben, den Benutzer zum Wiederholen der letzten Äußerung aufzufordern oder den Dialog durch einen menschlichen Operator weiterführen zu lassen. Die Variable Reference ist notwendig, um nach Füllern für Datenbankwächtern zu fragen, und nicht etwa nach Füllern der Dialogzielbeschreibungen. Mithilfe der Variablen ReferringExpressions können Dialogstrategien ausgedrückt werden, die im Falle ambiger Referenz Klärungsfragen generieren. Die SpeechAct Variable bestimmt die Struktur des Diskurses. Schließlich ist die Intention Variable notwendig, um den Gehalt der im Diskurs vorhandenen Information bezüglich der Dialogzielbeschreibungen zu charakterisieren und gegebenenfalls das Dialogsystem die Ausführung eines Dialogzieles bestätigen zu lassen. Damit erfüllen die Zustandsvariablen den Zweck, den Dialogzustand fein genug zu charakterisieren, daß eine genaue Beschreibung der Aktionen des Dialogsystems möglich ist.

Über die hier angegebenen Variablen hinaus ist es natürlich möglich, weitere Zustandsvariablen anzugeben, die den Dialogzustandsraum feiner unterteilen. Dies wird mit steigender Leistungsfähigkeit der dialogverarbeitenden Komponenten sicherlich nötig sein. So besteht in gegenwärtigen System keine Möglichkeit, Dialogzustände zu beschreiben, bei denen mithilfe von Counterfactuals verschiedene Alternativen exploriert werden.

Die Klassifizierung von Dialogzuständen ist auch in anderer Hinsicht interessant. Während in der vorliegenden Arbeit die Dialogstrategie symbolisch, nämlich durch ein constraint-logisches Programm angegegeben worden ist, kann man die Dialogstrategie auch erlernen. In der Arbeit von Walker et al [2002] beispielsweise wird der Dialogzustand durch eine Menge von 6 Merkmalen be-

schrieben. Die Dialogstrategie wurde mithilfe von $Q - Learning$ erlernt. Allerdings ist die dort beschriebene Charakterisierung des Dialogzustandes nicht domänenunabhängig. Mit anderen Worten, das Wechseln der Domäne würde erneut Datensammlung und Erlernen einer Dialogstrategie erfordern.

6.11 Zusammenfassung

In diesem Kapitel wurde beschrieben, wie der Zustand eines Dialogs zwischen Mensch und Maschine mithilfe von Zustandsvariablen abstrakt, das heißt sprach- und domänenunabhängig charakterisiert werden kann. Ein Dialogzustand kann dann angegeben werden durch die Variablenbelegung der Zustandsvariablen, wie zum Beispiel $\langle good, good, sa\_request, finalized, none, determined\rangle$. Mehrere Dialogzustände können durch die Angabe von *Don't Care* Elementen zusammengefaßt werden. So beschreibt zum Beispiel die partielle Variablenbelegung $\langle poor, \_, \_, \_, \_, \_\rangle$ alle Dialogzustände, in denen die Qualität der letzten Eingabe des Benutzers schlecht ist, ungeachtet des gegenwärtigen Zustandes bezüglich Dialogziele oder Datenbankanfragen.

Dieser Ansatz ist die Grundlage für die Entwicklung eines sprach- und domänenunabhängigen Dialogsystems. Durch die indirekte Beschreibung des Dialogzustandes durch die Zustandsvariablen wird ein *"layer of indirection"* eingeführt, der es möglich macht, den Fortschritt des Dialoges zu charakterisieren. Aktionen des Dialogsystems werden in Abhängigkeit partielle Belegungen der Zustandsvariablen ausgedrückt. Beispielsweise wird den durch $\langle poor, \_, \_, \_, \_, \_\rangle$ charakterisierten Dialogzuständen die Aktion zugeordnet, die den Benutzer zum Wiederholen der letzten Äußerung auffordert.

Im nächsten Kapitel wird geklärt, wie Aktionen des Dialogsystems charakterisiert werden können. Die darauffolgenden Kapitel ordnen dann konkrete Aktionen den Dialogzuständen zu.

Kapitel 7

Generische Interaktionsmuster

Im letzten Kapitel wurde beschrieben, wie der Zustand des fortschreitenden Dialogs durch die Angabe einiger weniger abstrakter Zustandsvariablen domänen- und zielsprachunabhängig beschrieben werden kann. Das Dialogsystem wird, nachdem es die Eingabe des Benutzers analysiert und verarbeitet hat, die Werte der Zustandsvariablen an den neuen Dialogzustand anpassen. Offensichtlich ist die Anpassung der Zustandsvariablen das informationsbasierte Äquivalent von Zustandsübergängen in Dialogsystemen, die auf endlichen Automaten basieren. Während im letzten Kapitel die Zustandsvariablen, das Äquivalent zu den Dialogzuständen, beschrieben worden ist, wird in diesem Kapitel beschrieben, wie die Verarbeitung der Eingabe und das Anpassen der Zustandsvariablen zustande kommt. Darüberhinaus wird beschrieben, wie sich die Diskursstruktur zu der Anpassung der Zustandsvariablen verhält.

Man könnte annehmen, daß ein Interaktionsmuster, definiert als ein Zustandsübergang zwischen abstrakten Dialogzuständen, einfach durch Start- und Endzustand charakterisiert werden kann. Damit wäre die Situation nicht sehr viel anders als im Falle endlicher Automaten. Wir erinnern uns jedoch, daß es die Aufgabe der abstrakten Klassifikation von Dialogzuständen war, den Fortschritt des Dialoges zu beschreiben. Als Konsequenz daraus kann der Folgezustand eines abstrakten Dialogzustandes nicht allein als Funktion des gegenwärtigen Zustandes und der Eingabe des Benutzers aufgefaßt werden. Vielmehr werden nach dem Erfassen der Eingabe gegebenenfalls verschiedene Verarbeitungsschritte ausgeführt. Diese schließen Datenbankzugriff, Anpassung des Diskursstruktur, Anforderungen von Diensten und so weiter mit ein. Der resultierende Dialogzustand wird dann aus einer Klassifikation nach dem Abschluß dieser Verarbeitungsschritte bestimmt. Dies geschieht wie im vorigen Kapitel beschrieben. Damit ergibt sich folgende Verarbeitungsabfolge:

1. Verarbeite Eingabe des Anwenders
2. Führe Datenbankanfragen, Dienstanforderungen etc aus wo immer möglich
3. Klassifiziere den gegenwärtigen Dialogzustand
4. gehe zu 1

Dadurch, daß in Schritt 2 externe Anwendungen direkt in die Manipulation der Repräsentationen im Diskurs mit eingebunden werden können, ergibt sich die Flexibilität des vorgeschlagenen Ansatzes.

7.1 Einleitung

Die Aufgabe eines natürlichsprachlichen Dialogsystems ist es, Benutzern eine Reihe von Diensten durch verbale Kommunikation zur Verfügung zu stellen. Die verfügbaren Dienste sind durch die Dialogzielbeschreibungen (siehe Abschnitt 6) repräsentiert. Dialog ist dann die Umsetzung des Bedürfnisses, einen Dienst auszuführen. Dialog-Management bestimmt, wie dieser Weg aussieht und ob der Benutzer an seinem Ziel ankommt.

Wenn man verschiedene existierende Dialogsysteme in verschiedenen Domänen betrachtet, können mehrere Gemeinsamkeiten festgestellt werden. Dialogsysteme haben Mechanismen, um nach fehlender Information zu fragen oder zwischen verschiedenen Optionen auszuwählen. Manche Systeme haben Mechanismen, die es erlauben, fehlerhafte und fehlerkannte Information zu bestätigen, bevor diese Information weiterverarbeitet wird. Andere Systeme wiederum haben Mechanismen, die es erlauben, im Diskurs etablierte Information durch Korrektur zurückzunehmen. Diese Mechanismen ähneln sich vom Prinzip der Verarbeitung, obwohl sie in verschiedenen Domänen, wie zum Beispiel Zugauskunft oder Hotelreservierung, angewandt werden können.

Das in diesem Kapitel vorgestellte Konzept des *Interaktionsmusters* erlaubt es, die Ähnlichkeiten der Interaktionsformen über die Domänen hinweg zu isolieren und wiederzuverwenden. Damit ähneln die Interaktionsmuster den aus dem *Software Engineering* bekannten *Entwurfsmustern* (engl. *Design Patterns*). Interaktionsmuster stellen eine Beziehung her zwischen (i) der Information und der Struktur des Diskurses, (ii) der Form des Austausches von Information zwischen Benutzer und Dialogsystem und (iii) der angeforderten Dienste der Anwendung.

7.2 Generische Interaktionsmuster

Während sich Interaktionsmuster den Entwurfsmustern insofern ähneln, als strukurelle Systemkomponenten für verschiedene Systeme wiederverwendet werden können, unterscheiden sie sich in den folgenden Punkten. Zunächst bestimmt das Vorhandensein oder die Abwesenheit eines Interaktionsmusters das Verhalten des Dialogsystems. Desweiteren werden Interaktionsmuster *instantiiert* und über Zeit entwickelt. Dies ist konform mit Ergebnissen von Forschungsarbeiten zu der Diskursstruktur [Grosz and Sidner, 1986].

Dialog-Management besteht aus der Auswahl und Instantiierung von Interaktionssequenzen zu gegebenen Zeitpunkten. Die Kriterien, nach denen geeignete Interaktionsmuster ausgewählt werden, sind in Abhängigkeit der abstrakten Dialogzustände formuliert.

Ein Interaktionsmuster beschreibt somit die Fähigkeit des Dialogsystems, eine spezifische Art von Informationsaustauch zwischen Benutzer und System zu verarbeiten. Die Instantiierung eines Interaktionsmusters beschreibt den konkreten Informationsaustausch, inklusive domänenabhängiger Repräsentationen. Dialog-Management besteht aus der geeigneten Auswahl und Instantiierung, d.h. Start und Fortführung, von Interaktionsmustern. Interaktionsmuster sind generisch, weil ein gegebenes Interaktionsmuster beschreibt, *welche* Information

ausgetauscht werden soll. Instantiierungen von Interaktionsmustern haben die Freiheit, den Austausch durch verschiedenen Mechanismen vorzunehmen. Mit anderen Worten, ein Interaktionsmuster beschreibt, *welche* Information ausgetaucht wird, aber nicht, *wie* diese Information ausgetauscht wird.

7.2.1 Eigenschaften der Interaktionsmuster

Bevor die Eigenschaften von Interaktionsmustern diskutiert werden können, müssen wir den Begriff des *Initiators* eines Interaktionsmusters einführen.

Definition 1 (Initiator eines Interaktionsmusters). *Der* Initiator I eines Interaktionsmusters *ist der Gesprächspartner, welcher den ersten Sprechakt des Interaktionsmusters mit der Absicht äußert, den Diskurs durch* $\mathbf{F}$ *und* $\mathbf{F}'$ *anzupassen. Der andere Gesprächspartner wird als* Opponent *bezeichnet.*

Wie oben ausgeführt, besteht ein Teil des Dialog-Managements unter Verwendung von Interaktionsmustern darin, aus dem vorhandenen Katalog von Interaktionsmuster eines auszuwählen, welches den Dialog näher zum Ziel bringt. Dies bedeutet aber, das Interaktionsmuster dadurch charakterisiert werden müssen, wie ihre Instantiierung die Information im Diskurs verändert. Dies folgt zur Postulierung der folgenden wünschenswerten Eigenschaft von Interaktionsmustern.

1. **Zielorientiertheit**
   Interaktionsmuster sind charakterisiert durch die Art und Weise, wie ihre Instantiierung Information im Diskurs anpaßt. Ist die bei Instantiierung des Interaktionsmusters gewünschte Änderung des Informationszustandes erreicht, gilt diese Instantiierung des Interaktionsmusters als *erfolgreich beendet.*

Ein Dialogsystem kann damit beispielsweise zu einem gegebenen Zeitpunkt im Dialog entscheiden, daß, um den Dialog zu beenden, Information vom Benutzer erfragt werden muß. Mit anderen Worten, Spezifizität der Information muß erhöht werden. Die Inspektion des Katalogs von Interaktionsmustern resultiert dann in einer Menge von Interaktionsmustern, deren Instantiierung die gewünschte Erhöhung der Spezifizität herbeiführen.

Damit sich die Interaktionsmuster charakterisiert werden können, gibt es eine Liste von Eigenschaften, die ein Interaktionsmuster charakterisieren. Diese Liste von Eigenschaften besteht in der vorliegenden Arbeit aus folgenden Attributen: (i) Initiierer des Interaktionsmuster mit den Werten *Benutzer* oder *System*, (ii) erlaubte Sprechakte des Initiierers, (iii) erlaubte Sprechakte des anderen Partners, (iv) zum Diskurs hinzuzufügende Information, und (v) vom Diskurs zu entfernende Information. Dialog-Management muß zu jedem Zeitpunkt in der Lage sein können,aufgrund dieser Eigenschaften ein Interaktionsmuster eindeutig auswählen zu können. Damit ergibt sich die letzte Eigenschaft von Interaktionsmustern.

2. **Heterogenität**
   In einem Katalog von Interaktionsmustern muß sich jedes Interaktionsmuster von allen anderen in mindestens einem der fünf charakterisierenden Merkmale unterscheiden.

Diese Eigenschaft ist nicht notwendigerweise einschränkend. Sind tatsächlich zwei Interaktionsmuster mit denselben Eigenschaften wünschenswert, kann die Forderung nach Heterogenität durch verschiedene Instantiierungen eines Interaktionsmusters umgangen werden.

Eine weitere wünschenswerte Eigenschaft ist, daß die Instantiierung eines Interaktionsmusters jederzeit abgebrochen werden kann. Damit wird vermieden, daß Benutzer von Dialogsystemen gezwungen werden, irrelevante Fragen zu beantworten, die nicht zu der Anforderung des vom Benutzer intendierten Dienstes beitragen. Diese Eigenschaft ist wie folgt beschrieben.

3. **Verletzbarkeit**
   Die Instantiierung eines Interaktionsmusters kann jederzeit beendet werden. Ist die bei Instantiierung des Interaktionsmusters gewünschte Änderung des Informationszustandes nicht erreicht, gilt diese Instantiierung des Interaktionsmusters als *erfolglos beendet*.

Desweiteren ist es wünschenswert, Interaktionsmuster verschachteln zu können. Wenn beispielsweise der Benutzer aus einer Liste von Optionen auszuwählen hat, soll ihm die Möglichkeit gegeben werden, mehr Information zu jeder dieser Optionen erfragen zu können. Diese rekursive Instantiierung von Interaktionsmustern führt zu folgender Eigenschaft.

4. **Unterbrechbarkeit**
   Der gegenwärtige Turn-Inhaber hat die Möglichkeit, ein neues Interaktionsmuster zu instantiieren, unabhängig davon, ob das zuletzt instantiierte Interaktionsmuster beendet worden ist oder nicht. Im letzteren Fall gilt das zuletzt instantiierte Interaktionsmuster als *unterbrochen*, nach Beendigung des gegenwärtigen Interaktionsmusters wird das zuletzt unterbrochene Interaktionsmuster als aktuelles Interaktionsmuster interpretiert.

Eine direkte Konsequenz aus der Unterbrechbarkeit von Interaktionsmustern ist die baumartige Diskursstruktur. Diese Eigenschaft ist in Übereinstimmung mit Ergebnissen aus der Diskursanalyse [Grosz and Sidner, 1986].

Wir bemerken, daß die Beobachtung des ersten Sprechaktes einer Instantiierung nicht notwendigerweise das instantiierte Interaktionsmuster bestimmt.

7.2.2 Definition von Interaktionsmustern

Aus der Menge der oben angegebenen Eigenschaften sind es die Eigenschaften (1) und (2), die die Komponenten von Interaktionsmustern bestimmen. Eigenschaften (3) bis (5) hingegen beschreiben die Instantiierung von Interaktionsmustern. Die Eigenschaften (1) und (2) führen dann zur folgenden Definition von Interaktionsmustern.

Definition 2 (Interactionsmuster). *Ein* Interactionsmuster *ist durch ein Tupel* $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}' \rangle$ *gegeben, wobei*

1. $I \subseteq \{system, user\}$ *die Menge erlaubten Initiatoren für dieses Interaktionsmuster,*

2. *SA_I die Menge der Sprechakte bezeichnet, die der Initiator dieses Interaktionsmuster zur Instantiierung dieses Interaktionsmusters verwenden darf,*
3. *SA_O die Menge der Sprechakte bezeichnet, die der Opponent des Initiators zur Instantiierung dieses Interaktionsmusters verwenden darf,*
4. *eine Menge $\mathbf{F}$ von Merkmalsstrukturen, die die Repräsentationen einschränkt, die dem Diskurs bei erfolgreichem Abschluß hinzugefügt wird,*
5. *eine Menge $\mathbf{F}'$ von Merkmalsstrukturen, die die Repräsentationen einschränkt, die aus dem Diskurs bei erfolgreichem Abschluß entfernt werden.*

7.3 Instantiierungen von Interaktionsmustern

7.3.1 Definition von Instantiierungen von Interaktionsmustern

Es ist wichtig, die Interaktionsmuster und deren Instantiierung zu unterscheiden. Ein Interaktionsmuster beschreibt, ob Information dem Diskurs hinzugefügt oder aus dem Diskurs entfernt werden soll. Die Instantiierung eines Interaktionsmusters beschreibt, wie dies geschehen soll.

Definition 3 (Sequenz von Turns). *Die* Sequenz von Turns *ist gegeben durch $\langle t_1, \ldots, t_n \rangle$, wobei ein Turn $t_i = \langle s_i, sa_i, F_i, F_i' \rangle$ gegeben ist durch einen Sprecher $s_i \in \{I, O\}$, den Sprechakt sa_i, einer Repräsentation F_i, die die dem Diskurs hinzuzufügende Information beschreibt, und einer Repräsentation F_i', die die aus dem Diskurs zu entfernende Information beschreibt.*

Eine Instantiierung eines Interaktionsmusters ist eine Sequenz von Turns, die die vom Interaktionsmuster geforderten Bedingungen erfüllt.

Definition 4 (Wohlgeformte Instantiierung). *Eine Sequenz von Turns $\langle t_1, \ldots, t_n \rangle$ ist eine* wohlgeformte Instantiierung *eines Interaktionsmusters $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}' \rangle$, wenn die folgenden Bedingungen gelten:*

1. **Initiator**
   Es gilt $s_1 \in I$.
2. **Wohlgeformtheit**
   Es gilt $sa_i \in SA_I$, wenn immer $s_i = I$ und $sa_i \in SA_O$, wenn immer $s_i = O$.
3. **Vollständigkeit**
   Bei Abschluß von der Instantiierung gilt $F \sqsubseteq \bigsqcup_i F_i$ und $F' \sqsubseteq \bigsqcup_i F_i'$', d.h. die bei Beginn der Instantiierung erwartete Informationsmenge wurde erbracht.

Eine Sequenz von Turns, für die die Bedingungen (1) bis (3) gelten, heißt partielle Instantiierung *des Interaktionsmusters.*

7.3.2 Heterogene Instantiierungen

Eine Instantiierung eines Interaktionsmuster ist eine Variante dessen, wie die vom Instantiierer erwartete Information erbracht werden kann. Es wurde in der Literatur [Sadek and De Mori, 1998] vermerkt, daß

> *... in natural communication, contextual interpretation is the general case rather than the exception, because ... the communicated message is always differential: a person does not describe a (whole) situation; she / he tends to express the difference between the situation she / he wants to describe and what she / he believes that her / his interlocutor already knows.*

In diesem Kontext können die Werte für $\mathbf{F}$ und $\mathbf{F}'$ als *Erwartungen* des Initiators des Interaktionsmusters betrachtet werden. Zum Beispiel, wenn der Initiator einer Interaktionsmuster die Frage stellt: *"Would you like to go to a French, Spanish or Mexican restaurant?"* dann ist die Erwartung $\mathbf{F}$ der Interaktionsmuster

$$\mathbf{F} = \{prp\_french, prp\_spanish, prp\_mexican\}$$

Da der Initiator der Frage nicht erwartet, daß Information aus dem Diskurs entfernt wird, gilt $\mathbf{F}' = \emptyset$. Dieses Beispiel illustriert, wie $\mathbf{F}$ und $\mathbf{F}'$ die Erwartungen an die *informationellen Unterschiede* repräsentieren, die eine erfolgreiche Terminierung eines Interaktionsmusters mit sich bringt.

Es ist eine Konsequenz der Separierung des Dialoges in Interaktionsmuster und deren Instantiierungen, daß die Spezifikation, *welche* Information durch eine Interaktion erbracht wird, getrennt ist von der Spezifikation, *wie* diese Information erbracht werden soll. Dies ist die Anwendung eines Grundsatzes *Separation of Concerns* des *Software Engineering*. Als ein Beispiel für verschiedene Interaktionsformen, vergleiche man, wie man eine Frage einem dreijährigen Kind, einem zwölfjährigen Kind oder einem Erwachsenen stellen würde. Während in allen drei Fällen die gewünschte Information dieselbe ist, werden die Formen der Fragen sich wohl unterscheiden. Dies entspricht drei verschiedenen Instantiierungen desselben Interaktionsmusters. Die unterschiedlichen Instantiierungen eines Interaktionsmusters bergen viele Vorteile in der Mensch-Maschine Kommunikation, wie beispielsweise dynamisches Anpassen an die Fähigkeiten des Spracherkenners.

Hinzu kommt, daß die verschiedenen Instantiierungen unterschiedliche Effekte auf den Anwender des Dialogsystems haben wird. Walker [1997] berichtet, daß sich Anwender sicherer im Umgang mit Dialogsystemen fühlen, wenn der Dialog gerichtet ist, und die Anwender genau instruiert werden, was sie sagen können. Auf der anderen Seite können Dialogsysteme nur unter Kenntnis der Interaktionsmuster entworfen werden. Die Instantiierungen können zu einem späteren Zeitpunkt variiert werden.

7.4 Instantiierungen von Interaktionsmuster und Dienstanforderungen

In Kapitel 5 wurde die Beziehung zwischen verfügbarer Information und unterstützten Dienstanforderungen beschrieben. Wie im obigen Abschnitt erläutert, stellt eine Instantiierung eines Interaktionsmusters die Beziehung zwischen dem fortschreitenden Dialog und den Veränderungen der Information im Diskurs her. Da sich somit das Informationspotential im Diskurs ändert, müssen

natürlich auch zusätzliche Dienste der Anwendung angefordert werden (falls $\mathbf{F} \neq \emptyset$) oder bereits angeforderte Dienste rückgängig gemacht werden (falls $\mathbf{F}' \neq \emptyset$). Somit kann der Zustand der Anwendung den Zustand des fortschreitenden Dialogs korrekt reflektieren.

7.4.1 Definition von unterstützten Diensten

Wie in den Kapiteln 2 und 5 beschrieben, werden in Abhängigkeit der Information im Diskurs Dienste der Anwendung angefordert. Die Dienstanforderungen, deren Informationsbedarf von der im Diskurs enthaltenen Information erfüllt wird, werden *unterstützte Dienstanforderungen* genannt.

Definition 5 (Unterstützte Dienstanforderungen). *Sei $\langle t_1, \ldots, t_k, t_{k+1} \rangle$ eine partielle Instantiierung eines Interaktionsmusters, G_k alle seit Beginn des Dialogs im Diskurs etablierte Information und $\mathbf{S}_k$ die Menge aller seit Beginn des Dialogs angeforderten Dienste. Die Mengen ΔS und $\Delta S'$ bezeichnen die nach Verarbeitung des $k+1$-ten Turn anzufordernden bzw. zu widerrufenden Dienste.*

7.5 Interaktionsmuster und Dialogzustände

In der Einleitung wurde vermerkt, daß die Instantiierung der Interaktionsmuster von der Funktion den Zustandsübergängen in einem auf endlichen Automaten basierenden Dialogsystem nahekommt. Im Kontext von abstrakten Dialogzuständen bedeutet dies, daß bei Abschluß der Instantiierung eines Interaktionsmusters die Repräsentation des Dialogzustandes angepaßt werden muß. Dies erlaubt dem Dialogsystem, Rückschlüsse auf den Fortschritt des Dialogs zu schließen. Ein Korrektur-Interaktionsmuster sollte beispielsweise die Variable, die die Qualität des fortschreitenden Dialogs repräsentiert, heruntersetzen. Auf der anderen Seite müssen Zustandsvariablen vom Dialogsystem neu angepaßt werden. Eine Instantiierung des Korrektur-Interaktionsmusters beispielsweise kann bewirken, daß durch bereits ausgeführte Datenbankzugriffe etablierte Referenzen wieder aus dem Diskurs entfernt werden. Damit muß der Wert der Variable REFERENCE von beispielsweise *ambiguous* auf *none* zurückgesetzt werden. Wir unterscheiden damit zwei Arten, auf die die Instantiierung eines Interaktionsmusters Einfluß zu die Repräsentation des Dialogzustandes hat.

7.6 Interaktionsmuster und Diskursstruktur

Wie in Abschnitt 5.2 beschrieben, muß Diskursstruktur notwendigerweise hierarchisch sein. Der einfache Algorithmus, der in der Einleitung angegeben war, berücksichtigt hierarchische Diskursstruktur jedoch nicht. Eine einfache Modifikation führt dann zu folgendem Algorithmus.

1. Erstelle semantische Repräsentation der Eingabe des Anwenders
2. (a) falls letzte Eingabe nicht Teil des gegenwärtigen Interaktionsmusters ist
      i. falls letzte Eingabe zum Abbruch des Interaktionsmusters führt, dann Instantiierung beenden

   ii. anderenfalls neues Interaktionsmuster bestimmen und in einem Unterdialogbaum instantiieren
3. Führe Datenbankanfragen, Dienstanforderungen etc aus wo immer möglich
4. Klassifiziere den gegenwärtigen Dialogzustand
5. gehe zu 1

7.7 Diskussion

Es gibt mehrere Ansätze für prinzipienbasiertes Dialog-Management in der Literatur. Einer der prominentesten Ansätze ist das von der France Télécom entwickelte Artimis System [Sadek and De Mori, 1998,Sadek *et al.*, 1997]. Dieses System stellt ein domänenunabhängiges, aus drei Komponenten bestehendes System dar. Die Komponenten sind ein Sprachverarbeitungskomponente, bestehend aus Spracherkenner und Parser, eine Generierungskomponente und die sogenannte *Rational Unit*. Diese *Rational Unit* ist eine auf der Modallogik KD45 basierende Inferenzmaschine, die aufgrund der vom Parser erzeugten Repräsentation und der Information im Diskurs mithilfe rationaler Prinzipien die nächste Aktion des Dialogsystems bestimmt. Die Axiome stellen sicher, daß sich das Dialogsystem "vernünftig" (engl. *rational*) verhält, also zum Beispiel dem Benutzer nicht Information mitteilt, die er schon kennt, oder umgekehrt, keine Information voraussetzt, die der Benutzer nicht kennen kann. Die Aktionen des Dialogsystems, sogenannte *communicative acts*, haben Vor- und Nachbedingungen, denenzufolge der Diskurs entsprechend angepaßt wird. Die hier eineführten Instantiierungen der Interaktionsmuster können als das funktionale Äquivalent der *communicative acts* betrachtet werden. Die Nachbedingungen der *communicative acts* im Falle von Artimis und die Werte für $\mathbf{F}$ und $\mathbf{F}'$ beschreiben beide, wie der Diskurs angepaßt werden muß. Als wesentliche Unterschiede zwischen den Ansätzen ist zu verbuchen, daß die Artimis Architektur keinerlei Aussagen über die Diskursstruktur macht und das die Einbindung von Anwendungsdiensten nicht so eng ist wie in der vorliegenden Arbeit.

7.8 Zusammenfassung

In diesem Kapitel wurde das Konzept des *Interaktionsmusters* eingeführt. Ein Interaktionsmuster beschreibt die wesentlich Fähigkeit eines Dialogsystems, Information in einer für das Interaktionsmuster typischen Art und Weise in den Diskurs zu integrieren, und den Zustand der Anwendung auf eine solche Art und Weise anzupassen, daß der Gesamtzustand des Systems konsistent bleibt. Interaktionsmuster können auf verschiedene Art und Weise instantiiert werden. Die Art der Instantiierung kann dabei in Abhängigkeit der Leistungsfähigkeit der Systemkomponenten wie zum Beispiel Spracherkenner angepaßt werden.

Interaktionsmuster isolieren die Mechanismen, die dem Informationsaustausch zwischen Dialogsystem und Anwender zugrunde liegen. Aus diesem Grunde bestimmt der Katalog der vom Dialogsystem unterstützten Interaktionsmuster das Verhalten, die Natürlichkeit und die Leistungsfähigkeit des Systems. In den folgenden drei Kapiteln werden verschiedene Interaktionsmuster vorgestellt.

Kapitel 8

Erfragen von Information

Eine der wichtigsten Funktionen von aufgabenorientierten Dialogsystemen ist es, Information vom Benutzer zu erfragen, um zu bestimmen, welchen Dienst des Systems der Benutzer in Anspruch nehmen möchte. In diesem Kapitel wird ein sprach- und domänenunabhängiges Interaktionsmuster entwickelt, das es dem Dialogsystem erlaubt, entsprechende Klärungsfragen zu generieren und deren Antworten in den Diskurs zu inkorporieren. Der Begriff der Klärungsfrage umfaßt im Rahmen dieser Arbeit die Begriffe Ja/Nein-Frage, Disjunktivfrage und Ersetzungsfrage.

Zunächst werden die abstrakten Dialogzustände (siehe Abschnitt 6.8) angegeben, in denen das Generieren einer Klärungsfrage erforderlich ist. Daraufhin wird beschrieben, wie der semantische Inhalt der Klärungsfrage bestimmt wird. Da die Dienste der Anwendung durch die Datenbankkonvertierungsbeschreibungen (siehe Abschnitt 5.3) und Dialogzielbeschreibungen (siehe Abschnitt 5.4) mit Hilfe von Merkmalsstrukturen beschrieben sind, kann ein sprach- und domänenunabhängiger Algorithmus zur Generierung von Klärungsfragen angegeben werden.

8.1 Einleitung

Die Aufgabe des Dialogsystems ist es, den vom Benutzer gewünschten Dienst und dessen Parameter zu bestimmen (siehe Abschnitt 6.7). Ist es dem Dialogsystem nicht möglich die Absicht des Benutzer eindeutig zu bestimmen, weil die Information im Diskurs nicht spezifisch genug ist, muß komplementäre Information interaktive erfragt werden. Zu diesem Zwecke wird ein Interaktionsmuster entwickelt, das es dem Dialogsystem erlaubt, entsprechende Fragen zu generieren und die Antwort in den Diskurs einzubringen. Das Interaktionsmuster wird demzufolge vom Dialogsystem initiiert (wir betrachten hier nicht den Fall, bei dem der Benutzer Klärungsfragen zu den Objekten unter Diskussion stellen kann).

Da es das Ziel dieses Interaktionsmusters ist, den Informationsgehalt zu erhöhen, fordern wir, daß $\mathbf{F} \neq \emptyset$ und $\mathbf{F}' = \emptyset$ gilt. Bevor das Interaktionsmuster formell definiert wird, werden die Fälle betrachtet, in denen die Anwendung des Interaktionsmusters notwendig ist.

8.2 Falluntersuchung

Im allgemeinen muß der Informationsgehalt des Diskurses erhöht werden, wenn das Dialogsystem nicht über genügend Information verfügt, um den nächsten anzufordernden Dienst eindeutig zu bestimmen. Hierbei wird unter *Dienst* entweder eine Datenbankanfrage (siehe Definition 1) oder ein anwendungsspezifischer Dienst (siehe Definition 5) verstanden. Mangelnde Spezifizität kann hierbei die folgenden drei Ursachen haben: (i) fehlende Information, (ii) mehrdeutige Information, und (iii) Informationmit zu geringer Konfidenzannotation durch den Spracherkenner.

Fehlende Information beschreibt die Situation, in der der Dialog nicht genügend fortgeschritten ist, um alle relevante Information erfragt zu haben. Dies ist beispielsweise der Fall, wenn der Benutzer den Zielort für eine Wegbeschreibung noch nicht angegeben hat. Im Falle ambiger Information sind mehr als eine Interpretation der bis jetzt akquirierten Information möglich, beispielsweise wenn der vom Benutzer verwendete Ausdruck des Zielortes auf mehrere Objekte referiert. Geringe Konfidenzraten sind auf Probleme in der Spracherkennungskomponente zurückzuführen und sollten bestätigt werden, bevor sie in den Diskurs integriert werden können.

8.2.1 Fehlende Information

Im Falle fehlender Information ist der Dialog nicht fortgeschritten genug, um die nächste Aktion des Dialogsystems eindeutig zu bestimmen. In der vorliegende Systemarchitektur können zwei verschiedene Fälle unterschieden werden. Im ersten Fall sind semantische Repräsentationen im Diskurs vorhanden, die durch Datenbankzugriffe vervollständigt werden könnten, aber keiner der entsprechenden Datenbankwächter (siehe Definition 4) ist erfüllt. Im zweiten Fall ist eine Dialogzielbeschreibung im Zustand *determined*, aber es ist nicht genügend Information vorhanden, um alle Parameter der der Beschreibung zugeordneten Dienste zu bestimmen (zur Erinnerung: eine Dialogzielbeschreibung ist *determined*, wenn die im Diskurs vorhandene Information nur mit dieser, aber keiner anderen Dialogzielbeschreibung konsistent ist, aber die vorhandene Information nicht ausreicht, um die Dialogzielbeschreibung zu erfüllen). Im folgenden werden die beiden Fälle genauer betrachtet.

Datenbankwächter. Datenbankwächter treten dann in Kraft, wenn die Information im Diskurs spezifisch genug ist, die Datenbank auszuwählen, aber die vorhandene Information nicht ausreicht, um die Einschränkung des Datenbankwächters zu erfüllen. Im Stadtinformationskiosk beispielsweise sind verschiedene Datenbanken für Restaurants, Hotels und so weiter vorhanden. Die Datenbankwächter für die Restaurantdatenbank sind in Abbildung 5.4 angegeben. Wenn die folgende atomare Merkmalsstruktur

$[obj\_restaurant]$

im Diskurs vorhanden ist, ist die Information ausreichend genug, um die Hoteldatenbank als Kandidaten auszuschließen, aber nicht ausreichend genug,

um den Datenbankzugriff durchzuführen. Diese Situation entspricht dem Zustand DatabaseSelected. Abhilfe schafft das Erfragen der Werte, die durch die Merkmalspfade der Datenbankwächter spezifiziert sind, also beispielsweise Preiskategorie oder Nationalität.

Dialogzielbeschreibungen. Dialogzielbeschreibungen sind, ähnlich wie Datenbankwächter, Schranken für Informationsgehalt. Analog zu Datenbankwächtern, kann fehlende Information die Generierung einer Ersetzungsfrage erfordern. Die Situation ist dieselbe wie im Falle der Datenbankwächter: eine Dialogzielbeschreibung wurde ausgewählt, aber die Information im Diskurs reicht nicht aus, die Beschreibung zu erfüllen.

Das Generieren einer Ersetzungsfrage ist nur dann erforderlich, wenn die Zustandsveriable Intention den Wert *determined* hat. Um zu sehen, warum, nehmen wir an, daß die Variable einen Wert ungleich determined hat. Für *neutral, inconsistent* und finalized stellt sich die Notwendigkeit einer Frage nicht, da, respektive, entweder der Dialog noch nicht initiiert ist, die Aussagen des Benutzers falschverstanden oder inkonsistent mit dem Aufgabenmodel sind oder das Dialogziel erreicht ist. Bleibt der Fall Intention = *selected*. In diesem Fall muß bestimmt werden, welches der ausgewählten Dialogziele der Benutzer verfolgen möchte. Dies kann jedoch am besten durch eine Disjunktivfrage geschehen, wobei Referenzen zu den ausgewählten Dialogzielbeschreibungen die Disjunkte darstellen.

Wir können also annehmen, daß Dialogzielbeschreibungen nur dann die Ursache für Ersetzungsfragen sein können, wenn die Zustandsvariable den Wert *determined* hat. Dies impliziert aber, daß das vom Benutzer intendierte Dialogziel bekannt ist. Wenn in der Dialogzielbeschreibung Merkmalspfade angegeben sind, die im Diskurs entweder nicht definiert sind oder deren Wert weniger spezifisch als der entsprechende Wert in der Dialogzielbeschreibung ist, muß dieser Wert durch eine Ersetzungsfrage erfragt werden.

8.2.2 Ambige Information

Im Gegensatz zu den Ersetzungsfragen, die im Falle fehlender Information angewandt werden, können Disjunktivfragen den Benutzer aus mehreren Alternativen auswählen lassen. Listen von Alternativen können zu verschiedenen Zeiten im Dialog generiert werden. In der vorliegenden Architektur werden Listen von Alternativen durch nicht eindeutig bestimmte Dialogzielbeschreibungen, durch Datenbankzugriffe selbst und Parse n best Listen erzeugt.

Ambige Information aus Datenbankzugriffen. Nachdem die Datenbankwächter durch referierende Ausdrücke erfüllt worden sind, wird der Datenbankzugriff vorgenommen und Repräsentationen der Objekte, auf die referiert wird, in den Diskurs integriert. Hierbei kann es vorkommen, daß referierende Ausdrücke mehr Referenzen haben, als die Dialogzielbeschreibungen, denen sie übergeben werden, erlauben. Zur Erinnerung: Die Dialogzielbeschreibung ermöglichen durch die Angabe von *min* und *max* Constraints die Anzahl der

Repräsentationen, für die die assoziierten Dienste angefordert werden, zu beschränken; siehe auch die Definitionen in Abschnitt 5.4.1. In diesem Fall muß zusätzliche Information erfragt werden, die die Menge der Objekte, auf die referiert werden kann, verringert.

Aus Darstellungsgründen wird zunächst angenommen, daß dieser Fall nur auftritt, wenn die Zustandsvariable INTENTION den Wert *Selected* hat. Mit anderen Worten, es wird angenommen, daß das Dialogsystem bestimmt hat, welche Aufgabe der Benutzer ausführen will, muß aber noch die Ergebnismenge der Datenbankanfrage reduzieren (für Verarbeitung ambiger Referenz in Situationen, in denen keine Dialogzielbeschreibung im Zustand *Selected* ist, siehe den nächsten Abschnitt). Nehmen wir an, daß eine semantische Repräsentation der Äußerung im Diskurs repräsentiert ist. Als Beispiel dient der Satz *"Give me directions to a restaurant in Squirrel Hill"* und seine semantische Repräsentation

$$\begin{bmatrix} act\_givedirections \\ \text{ARG} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant \\ \text{NEIGHBORHOOD } \textit{"Squirrel Hill"} \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{8.1}$$

Der Wert des Merkmalpfades ARG ist die Substruktur, die eine definite Beschreibung repräsentiert. Datenbankzugriff und anschließende Umwandlung in Merkmalsstrukturen der Ergebnismenge, wie in Abschnitt 5.3 beschrieben, führt zu einer unterspezifizierten Repräsentation im Diskurs. Wir betrachten nun zunächst den Wert der Merkmalpfades ARG|DST|NATIONALITY, da die Disambiguierungsalgorithmen der Basis-Dialogdienste (siehe Abschnitt 4.5) den Wert diesen Pfades am effizientesten disambiguieren können. Der unterspezifizierte Knoten dieses Merkmalpfades hat folgende Struktur:

$$nationality^{(1,\ldots,19)} \begin{cases} \underline{american}^{(1,\ldots,9)} \\ \underline{asian}^{(10,\ldots,14)} & \begin{cases} chinese^{(10,\ldots,12)} \\ japanese^{(13)} \\ thai^{(14)} \end{cases} \\ european^{(15,\ldots,18)} & \begin{cases} \underline{greek}^{(15,16)} \\ \underline{italian}^{(17,18)} \end{cases} \\ \underline{mediterranean}^{(19)} \end{cases} \tag{8.2}$$

Die Aussage der Repräsentation ist, daß es neun amerikanische, fünf asiatische, drei chinesische usw. Restaurants gibt, die die Constraints des Benutzers erfüllen, in Squirrel Hill zu sein. Die einfachste und effizienteste Weise, die Nationalität des gewünschten Restaurants zu bestimmen wäre, alle vorhandenen Nationalitäten aufzuzählen und den Benutzer auswählen zu lassen.[1] Allerdings ist deren Zahl zu hoch als daß es für eine auditorische Präsentation durch ein

[1] Dies wird allerdings nur in manchen Fällen den Zielort eindeutig bestimmen. Wählt der Benutzer beispielsweise ein chinesisches Restaurant, muß zwischen den verbleibenden drei chinesischen Restaurants disambiguiert werden.

Sprachsynthesesystem praktikabel wäre. Deswegen wählt das System eine Untermenge aus, die eine gewisse vorgegebene Höchstschranke nicht übersteigt, im vorliegenden System 5. Wie dies genau geschieht, ist ebenfalls im Abschnitt 4.5 beschrieben. Im diesem Beispiel ist die Auswahl durch die unterstrichenen Knoten in (8.2) gegeben, die die Menge der vorhandenen Nationalitäten partitioniert. Dieser Knoten wird dann mithilfe der Generierungsschablonen (siehe Abschnitt 4.5 in eine Disjunktivfrage der Form *"Would you like to go to an American, Asian, Greek, Italian or mediterranean place?"* umgewandelt. Wählt der Benutzer ein asiatisches Restaurant, wird der unterspezifizierte Knoten in (8.2) zu dem Knoten

$$\underline{asian}^{(10,\ldots,14)} \begin{cases} chinese^{(10,\ldots,12)} \\ japanese^{(13)} \\ thai^{(14)} \end{cases} \tag{8.3}$$

reduziert. Da der reduzierte Knoten in (8.3) nach wie vor der Knoten ist, der die verbleibende Menge an Restaurants am effizientesten disambiguiert, wird dem Benutzer nun die Gelegenheit gegeben, zwischen chinesischen, japanischen und thailändischen Restaurants auszuwählen. Wählt der Benutzer nun chinesische Restaurants aus, ist die Nationalität des Zielortes eindeutig bestimmt, und ein anderes Unterscheidungsmerkmal muß zur Disambiguierung herangezogen werden.

Ambige Intention. Das im vorhergehenden Abschnitt dargestellte Verfahren beruht auf der Annahme, daß die Zustandsvariable INTENTION durch den Wert *determined* belegt ist. Nun wird untersucht, wie in anderen Dialogzuständen zu verfahren ist. Nehmen wir also an der Wert der Variablen sei *Inconsistent*. In diesem Fall ist die im Diskurs vorhandene Information inkonsistent mit den Dialogzielbeschreibungen, und eine Klärungsfrage wird nichts zum Fortschreiten des Dialogs beitragen. Deswegen kann in diesem Dialogzustand keine Klärungsfrage generiert werden. Sei nun der Wert der Variablen *Finalized*. In diesem Zustand wurde das Dialogziel des Dialoges bereits erfüllt, und Klärungsfragen sind überflüssig. Ist der Wert der Variablen hingegen *Neutral*, dann wurde der Dialog noch nicht gestartet, und die Generierung von Klärungsfragen ist ebenfalls nicht möglich. Im Gegensatz dazu ist für den Wert *Selected* das vom Benutzer intendierte Dialogziel nicht eindeutig bestimmt, und eine entsprechende Klärungsfrage ist angebracht. Wir können also festhalten, daß Klärungsfragen nur dann generiert werden müssen, wenn die Variable INTENTION entweder den Wert *Determined* oder *Selected* annimmt.

Wir betrachten nun den Fall, in dem die Variable INTENTION den Wert *Selected* annimmt. In diesem Fall gibt es mehr als eine Dialogzielbeschreibung, die mit der Information im Diskurs kompatibel ist. Ein einfaches Verfahren, den Benutzer aus diesen n Dialogzielbeschreibungen die intendierte auswählen zu lassen, ist, die Merkmalsstrukturen D der Dialogzielbeschreibungen als Basis für die Generierung von unterspezifizierten Merkmalsstrukturen zu nehmen. Auf die so generierte unterspezifizierte Merkmalsstruktur kann dann das im vorigen Abschnitt beschriebene Verfahren zur Disambiguierung angewandt werden. Dies wird solange wiederholt, bis nur noch eine Dialogzielbeschreibung

kompatibel mit der Information im Diskurs ist. Damit wechselt der Wert der Variable INTENTION von *Selected* zu *Determined*, und das im vorigen Abschnitt beschriebene Verfahren disambiguiert die Repräsentationen, falls nötig.

Wir vermerken, daß das Disambiguieren von Dialogzielbeschreibungen nur deswegen funktioniert, weil wir annehmen können, daß die Merkmalsstrukturen der Dialogzielbeschreibungen miteinander inkompatibel sind. Dies ist garantiert durch die *Independent Goal Condition* des Aufgabenmodells (siehe Definition 7). Nehmen wir um der Diskussion willen an, daß die *Independent Goal Condition* verletzt ist. Dies bedeutet, daß es zwei Dialogzielbeschreibungen $\mathbf{G}_1$ und $\mathbf{G}_2$ mit zugeordneten Merkmalsstrukturen D_1 und D_2 gibt, so daß entweder $D_1 \sqsubseteq D_2$ oder $D_2 \sqsubseteq D_1$. Nehmen wir o.B.d.A an, daß $D_1 \sqsubseteq D_2$ gilt. Sei nun $D = D_1$ die im Diskurs vorhandene Information. Dies impliziert nach Definition 7, daß der Zustand von $\mathbf{G}_1$ und $\mathbf{G}_2$ SELECTED ist. Sollte das Hinzufügen von Information durch Klärungsfragen zum Diskurs $\mathbf{G}_1$ deselektieren, bedeutet dies, daß die im Diskurs vorhandene Information mit D_1 inkompatibel ist. Da aber $D_1 \sqsubseteq D_2$ gilt, ist auch D_2 mit der im Diskurs vorhandenen Information inkompatibel und $\mathbf{G}_2$ wird ebenfalls deselektiert. Dies zeigt, daß bei Verletzen der *Independent Goal Condition* die Zustandsübergänge von Dialogzielbeschreibungen nicht mehr unabhängig voneinander sind. Dies ist jedoch kein größeres Problem in praktischen Anwendungen, da die *Independent Goal Condition* zum Übersetzungszeitpunkt des Systems überprüft und entsprechende Korrekturen in den Repräsentationen einfach vorgenommen werden können.

8.2.3 Geringe Konfidenzannotation

Die semantischen Repräsentationen sind mit Konfidenzannotationen des Spracherkenners versehen. Somit ist es möglich, selektiv Information zu ignorieren oder bestätigen zu lassen, sollte die Konfidenzannotation wahrscheinliche Fehlerkennung anzeigen. Die Repräsentation

$$\begin{bmatrix} \langle act\_givedirections, conf\_high \rangle \\ \text{ARG} \begin{bmatrix} \langle obj\_path, conf\_high \rangle \\ \text{DST } \langle obj\_bar, conf\_low \rangle \end{bmatrix} \\ \\ \end{bmatrix} \tag{8.4}$$

zeigt eine Repräsentation des Satzes *"Give me directions to a bar"*, wobei das Wort *bar* eine niedrige Konfidenzannotation zugewiesen bekommen hat. [2]

8.2.4 Zusammenfassung der Falluntersuchung

Die oben beschriebenen Situationen stellen Systemzustände dar, in denen die Generierung einer Klärungsfrage den Dialog vorantreibt. Die Systemzustände lassen sich nach Art der zu stellenden Frage klassifizieren. Tabelle 8.2.4 zeigt die Abbildung von Systemzuständen auf zu generierende Frageformen. Um die richtige Frageform auszuwählen, wird nicht nur der Systemzustand, sondern auch nähere Umstände wie Grad der Ambiguität in Betracht gezogen.

[2] Konfidenzannotationen sind in den anderen Darstellungen der Repräsentationen der Übersichtlichkeit halber nicht angegeben.

Systemzustand	Frageform	Bedeutung
$\langle poorQuality, \_, \_, \_, \_\rangle$	Ja/Nein Frage	Bestätigung unsicherer Information
$\langle \_, \_, Selected, \_, \_\rangle$	Ersetzungsfrage Disjunktivfrage	Erfragen fehlender Information für Datenbankzugriff
$\langle \_, \_, \_, \_, AmbiguousReference, \_\rangle$	Disjunktivefrage	Disambiguierung ambiger Referenzen
$\langle \_, \_, \_, \_, \_, Selected\rangle$	Disjunktivefrage	Disambiguierung mehrerer Ziele
$\langle \_, \_, \_, \_, \_, Determined\rangle$	Disjunktivefrage Ersetzungsfrage	Erfragen fehlender Information für die gewählte Dialogzielbeschreibung

Tabelle 8.1. Die Klassifikation der Systemzustände, in denen die Generierung einer Klärungsfrage angebracht ist

8.3 Das Fragen Interaktionsmuster

In den oben beschriebenen Situationen muß zusätzliche Information in den Diskurs eingebracht werden, um den Dialog fortsetzen zu können. Dies geschieht mithilfe des FRAGEN-Interaktionsmusters. Da es Aufgabe dieses Interaktionsmusters ist, die Spezifizität der Repräsentationen zu erhöhen, wird $\mathbf{F} \neq \emptyset$ und $\mathbf{F}' = \emptyset$ gefordert. Fragen können vom System wie auch vom Benutzer initiiert werden. Die Beschränkung der Sprechakte auf $SA_I = \{act\_question\}$, $SA_O = \{act\_inform\}$ ergibt sich aus der Tatsache, daß der Initiierer des Interaktionsmusters solange fragt, bis die gewünschte Information im Diskurs etabliert ist, und der Opponent entsprechend antwortet. Damit ergibt sich folgende Definition.

Definition 1 (Klärungsfragen-Interaktionsmuster). *Ein* Klärungsfragen-Interaktionsmuster *ist ein Interaktionsmuster* $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}', \rangle$, *wobei* $SA_I = \{act\_question\}$, $SA_O = \{act\_answer\}$, $\mathbf{F}$ *entspricht dem semantischen Inhalt der erwarteten Antworten und* $\mathbf{F}' = \emptyset$.

Bevor verschiedene Instantiierungen des Interaktionsmusters betrachtet werden, soll noch auf die Verwendung der Bezeichnungen *Disjunktivfrage* und *Ersetzungsfrage* hingewiesen werden. Vom Standpunkt der Dialogverarbeitung ist die Eigenschaft der Disjunktivfrage wichtig, dem Benutzer die verfügbaren Optionen mitzuteilen. Der zweite Satz in *"There are Chinese restaurants in Squirrel Hill, Shadyside and East Liberty. Which neighborhood would you like to go to?"* ist zwar technisch gesehen eine Ersetzungsfrage. Da jedoch der gesamte Ausdruck vom Standpunkt der Dialogverarbeitung äquivalent zur Disjunktivfrage *"Would you like to go to a Chinese restaurant in Squirrel Hill, Shadyside or East Liberty?"* ist, wird der Ausdruck in beiden Fällen als Disjunktivfrage bezeichnet.

8.3.1 Instantiierung mittels Disjunktivfragen

Abbildung 8.1 zeigt ein Beispieldialog mit mehreren Instantiierungen des FRAGEN-Interaktionsmusters. Alle Instantiierungen benutzen Disjunktivfragen.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a restaurant in Squirrel Hill	Qst IS
System:	act_inform	There are 19 places in Squirrel Hill.	Question IS
	act_request	Would you like to go to an Asian, American, Greek, Italian or Mediterranean place?	
User:	act_inform	Asian	
System:	act_inform	There are five Asian places in Squirrel Hill.	
	act_request	Would you like to go to a Chinese, Thai or Japanese place?	
User:	act_inform	Chinese	
System:	act_inform	There are three Chinese places in Squirrel Hill.	Qst IS
	act_request	Would you like to go to a cheap or moderately expensive place?	
User:	act_inform	Moderately expensive.	
System:	act_inform	I am giving you directions to 'Silk Pagoda' in Squirrel Hill.	

Abbildung 8.1. Eine Instantiierung des Fragen-Interaktionsmusters mit Disjunktivfragen.

8.3.2 Instantiierung mittels Ersetzungsfragen

Abbildung 8.2 zeigt ein Beispieldialog mit mehreren Instantiierungen des FRAGEN-Interaktionsmusters. Die zweite Instantiierung des FRAGEN-Interaktionsmusters benutzt eine Ersetzungsfrage.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a restaurant in Squirrel Hill	Qst IS
System:	act_inform	I am not sure I understood everything you said.	
	act_request	Which neighborhood would you like to go to?	Qst IS
User:	act_inform	Squirrel Hill.	
System:	act_inform	There are 19 places in Squirrel Hill.	
	act_request	Would you like to go to Asian, American, Italian, Greek or Mediterranean place?	Qst IS
User::	act_inform	Italian	
System:	act_inform	I am giving you directions to 'Sweet Basil' in Squirrel Hill.	

Abbildung 8.2. Eine Instantiierung des Fragen-Interaktionsmusters mit Ersetzungsfragen.

8.4 Implementierung

8.4.1 Algorithmus

Entscheidet das Dialogsystem, daß das FRAGEN-Interaktionsmuster instantiiert werden muß, werden alle dem System bekannten Schablonen vom Typ *enum-qst,infoqst* oder *yesnoqst*, deren Constraints erfüllt sind, wird instantiiert, und die resultierende Äußerung wird dem Benutzer präsentiert.

8.4.2 Variabilität in der Fragestellung

In der Vergangenheit wurde gezeigt, daß Benutzer von natürlichsprachlichen Dialogsystemen es vorziehen, durch den Dialog "geführt" zu werden [Walker *et al.*, 1997]. Unzuverlässige Spracherkenner oder fehlende Grammatikabdeckung können durch spezifisch formulierte Fragen so gut wie ausgeschlossen werden. Für das hier vorgestellte System bedeutet dies, daß Disjunktivfragen den offeneren Ersetzungsfragen vorgezogen werden und Ja/Nein-Fragen den Disjunktivfragen vorgezogen werden.

Es ist in begrenztem Maß möglich, die Form der Fragen ineinander umzuwandeln. Unter welchen Umständen dies geschehen kann, wird in diesem Abschnitt beschrieben.

Umwandeln einer Ersetzungsfrage in eine Disjunktivfrage. Ersetzungsfragen erfragen fehlende Information in einer Repräsentation. Eine typische Situation, in der Ersetzungsfragen nötig sind, ist bei fehlender Information in einer Dialogzielbeschreibung **G**, die sich im Zustand DETERMINED befindet. Diese fehlende Information kann beispielsweise die gewünschte Preiskategorie eines Hotelzimmers sein.

Dazu nehmen wir an, daß die Repräsentation (8.5) im Diskurs enthalten ist.

$$\begin{bmatrix} act\_reservehotel \\ \text{ARG } obj\_hotel \end{bmatrix} \tag{8.5}$$

Die Merkmalsstruktur der Dialogzielbeschreibung **G** [3] ist durch (8.6) gegeben.

$$\begin{bmatrix} act\_reservehotel \\ \text{ARG} \begin{bmatrix} obj\_hotel \\ \text{PRICECATEGORY } prp\_price \end{bmatrix} \end{bmatrix} \tag{8.6}$$

Die entsprechende Ersetzungsfrage könnte etwa lauten: *"Welche Preiskategorie wünschen Sie?"*.

Disjunktivfragen auf der anderen Seite erlauben dem Benutzer, aus einer Liste von Optionen auszuwählen. Dies ist besonders in Situationen nach Datenbankzugriffen mit großen Ergebnismengen der Fall, wie es im obigen Beispiel des Restaurants beschrieben wurde. Im Falle fehlender Information ist eine solche Liste von Optionen jedoch nicht erhältlich. Da jedoch die Typenhierarchie Constraints über die erlaubten Werte eines Merkmalspfades bereitstellt, ist es möglich, eine Liste von Optionen aus der Typenhierarchie abzuleiten. Wenn es beispielweise drei Preiskategorien für Hotelzimmer gibt, repräsentiert durch die Typen $prp\_cheap, prp\_moderate, prp\_expensive$, dann kann die Liste von Optionen bestimmt werden durch die Menge von Typen τ der Typenhierarchie,

[3] Die Dialogzielbeschreibung ist für dieses Beispiel vereinfacht worden; so sind zum Beispiel Merkmals für Ankunfts- und Abfahrtsdatum nicht gezeigt.

für die $prp\_price \subset \tau$ gilt. In diesem Fall ist dies

$$prp\_price \sqsubseteq prp\_cheap$$
$$prp\_price \sqsubseteq prp\_medium$$
$$prp\_price \sqsubseteq prp\_expensive$$

Basierend auf dieser Information, wird ein unterspezifizierter Knoten (8.7) generiert,

$$prp\_price^{(1-3)} \begin{cases} prp\_cheap^{(1)} \\ prp\_medium^{(2)} \\ prp\_expensive^{(3)} \end{cases} \tag{8.7}$$

der, mithilfe von Kontextinformation 8.8

$$\begin{bmatrix} act\_reservehotel^{(1-3)} \\ \text{ARG} \begin{bmatrix} obj\_hotel^{(1-3)} \\ \\ \text{PRICECATEGORY } prp\_price^{(1-3)} \begin{cases} prp\_cheap^{(1)} \\ prp\_medium^{(2)} \\ prp\_expensive^{(3)} \end{cases} \end{bmatrix} \end{bmatrix} \tag{8.8}$$

zur Generierung der Disjunktivfrage *"Would you like to make a reservation in a cheap, moderately expensive or expensive hotel?"* herangezogen wird. Dieses Verfahren funktioniert auch mit mehr als 5 Optionen, solange die Optionen durch zusätzliche Typen hierarchisch angeordnet sind. Dies ist analog zu den Typen *european* und *asian* im obigen Beispiel.

Umwandeln von Disjunktivfragen in Ja/Nein Fragen. Eine Disjunktivfrage mit n Optionen kann einfach in maximal n äquivalente Ja/Nein Fragen umgewandelt werden. Dazu werden Fragen der Form *"Wünschen Sie* $langleoption_i\rangle$ generiert, bis der Benutzer die Frage positiv beantwortet.

Umwandeln von Ja/Nein Fragen in Disjunktivfragen. Umwandlungen in die andere Richtung sind theoretisch möglich, werden aber vom System in der vorliegenden Implementierung nicht vorgenommen.

Umwandeln von Disjunktivfragen in Ersetzungsfragen. Ebenfalls sind Umwandlungen von Disjunktivfragen in Ersetzungsfragen theoretisch möglich, werden aber vom System in der vorliegenden Implementierung nicht vorgenommen.

8.4.3 Verweigerung der Antwort

Es ist möglich, daß der Benutzer die Antwort auf eine Frage nicht kennt. Eine Frage nach Stadtteilen beispielsweise wird ein ortsfremder Benutzer nicht beantworten können. Aus diesem Grunde ist es möglich, daß der Anwender mit

dem Ausdruck *"I don't know it!"* die Antwort verweigert. Intern wird der Merkmalspfad, nach dessen Wert gefragt worden ist, mit einem Index versehen, der anzeigt, daß der Benutzer die Antwort auf diese Frage nicht weiß. Schablonen, die nach mit dem Index versehenen Merkmalspfaden fragen, werden dann bei der Auswahl ignoriert. Kommt es dann vor, daß das System keine weiteren Fragen mehr generieren kann, weil der Anwender die Antworten nicht weiß, wird ein Objekt zufällig ausgewählt, um einen Abbruch des Dialogs zu vermeiden.

8.4.4 Integration mit dem Spracherkenner

Um die Spracherkennung robuster zu gestalten, wird das Sprachmodell des Spracherkenner bei der Erkennung der Antwort eingeschränkt. Es ist dann nur erlaubt, eine der dem System bekannten Antworten zu nennen.

8.5 Diskussion

Viel der hier beschriebenen Verfahren ähnelt früheren Forschungsarbeiten in Dialogsystemen, so zum Beispiel [Ward, 1994,Papineni *et al.*, 1999,Abella and Gorin, 1999]. Es sind jedoch zwei wesentliche Unterschiede zwischen früheren Arbeiten und den hier vorgestellten Ansätzen festzuhalten. Zunächst ist der binäre Begriff von vorhandener oder nicht vorhandener Information dank der Subsumptionsrelation der Merkmalsstrukturen verfeinert worden. In konventionellen *slot-filling* Ansätzen kann über fehlende Werte keine Aussage gemacht werden außer daß sie fehlen. Im hier verfolgten Ansatz kann man jedoch dank der Typinformation beispielsweise Aussagen über die Anzahl möglicher Antworten machen und damit, wie in Abschnitt 8.4.2 erläutert, eventuell besser geeignete Fragen gestellt werden. Im obigen Beispiel des Hotelpreises wären mögliche Füller prp_cheap, prp_medium oder $prp\_expensive$, und die Frage *"Was für ein Hotelzimmer möchten Sie?"* kann automatisch umgewandelt werden in *"Möchten Sie ein preiswertes, komfortables oder luxuriöses Hotelzimmer?"*. Da im zweiten Fall die möglichen Optionen aufgelistet sind, ist die Benutzerführung besser und die Anforderungen an den Spracherkenners sind nicht so hoch.

Der zweite und wichtigere Aspekt der hier verfolgten Arbeit ist jedoch, daß ein *"layer of indirection"* besteht zwischen dem Dialogzustand und der Aktion, die das Dialogsystem auszuführen hat. Damit können Dialogstrategien unabhängig von der Domäne oder Zielsprache variiert werden. Dies ist eine Eigenschaft, die dieses Interaktionsmuster mit den in den folgenden Kapiteln vorgestellten Interaktionsmustern teilt.

8.6 Zusammenfassung

In diesem Kapitel wurden verschiedene Dialogzustände beschrieben, in denen Klärungsfragen notwendig sind. Die Ausführung von Klärungsfragen wird von einem Interaktionsmuster übernommen.

Das Klärungsfragen-Interaktionsmuster verdeckt die konkrete Dialogstrategie, mit der die fehlende Information vom Benutzer akquiriert wird. In dem

Moment, wo – durch Untersuchung der abstrakten Dialogzustände festgestellt wird – daß Information akquiriert werden muß, wird das Klärungsfragen- Interaktionsmuster instantiiert. Dabei wird spezifiziert, welche Information zur Weiterführung des Dialoges benötigt wird. Zu entscheiden, wie diese Information nun vom Benutzer akquiriert wird, sei es durch Ersetzungsfragen, sei es Disjunktivfragen, obliegt dabei der Implementierung des Interaktionsmusters.

Das Klärungsfragen-Interaktionsmuster ist das Paradebeispiel, das zeigt, wie machtvoll die Trennung von Dialogstrategie (*wie* Information akquiriert wird) von Dialogverstehen auf höherer Ebene (*was* das Dialogsystem als nächstes tun soll) ist. Dies liegt daran, daß Implementierungen des Klärungsfragen-Interaktionsmuster über so viele Freiheitsgrade verfügen.

Die in den folgenden Kapiteln beschriebenen Interaktionsmuster folgen demselben Prinzipien von Trennung der Dialogstrategie von Dialogverstehen auf höherer Ebene.

Kapitel 9

Zurücknahme von fehlinterpretierter Information

In menschlicher Kommunikation können auftretende Mißverständnisse schnell und effizient behoben werden. Personen können interaktiv Mißverständnisse entdecken und beheben. Diese Fähigkeit ist wesentlich für den Erfolg von effizienter Kommunikation.

Im Falle natürlichsprachlicher Dialogsysteme ist die Notwendigkeit von Algorithmen zur Behebung von Mißverständnissen noch dringender, da sowohl die Komponenten für die Spracherkennung als auch für die Satzanalyse in der Regel nicht fehlerfrei arbeiten. Allerdings ist der Entwurf und die Implementierung effektiver Algorithmen zur Behebung von Mißverständnissen überraschend schwierig. Um zu sehen, wo die Schwierigkeiten liegen, muß man die Gründe der auftretenden Mißverständnisse untersuchen. Die zur Zeit von natürlichsprachlichen Dialogsystemen abgedeckten Domänen sind einfach genug, um semantische Mißverständnisse auszuschließen: Im Falle eines Zugauskunftssystems "verstehen" die Konversationspartner die Bedeutung des Wortes "Abfahrtsbahnhofs", beispielsweise. Mißverständnisse treten demzufolge meist durch unzulängliche Spracherkenner- und Satzanalysetechnologie auf. Diese Unzulänglichkeiten sind meist die Folge fehlender Abdeckung des Problembereichs auf drei verschiedenen Ebenen, nämlich (i) lexikalisch, (ii) strukturell und (iii) semantisch. Auf lexikalischer Ebene tritt das *Out-of-Vocabulary* Problem auf, wenn Worte verwendet werden, die im Vokabular des Erkenners nicht vorhanden sind. Dies führt unweigerlich zu Fehlerkennungen und damit zu einer semantischen Repräsentation, die die intendierte Bedeutung nicht korrekt wiedergibt. Auf struktureller Ebene treten Probleme auf, wenn bekannte Worte in einer Konfiguration verwandt werden, die dem Spracherkenner oder dem Parser nicht bekannt sind. Sprachmodellbasierte Erkenner haben hier Vorteile gegenüber grammatikbasierten Erkennern. falls genügend große Trainingsmengen vorhanden sind. Desweiteren kompensieren die Skip-Mechanismen robuster semantischer Parser für fehlende Abdeckung, sind dann aber nicht immer in der Lage, die intendierte Bedeutung der Äußerung zu erfassen. Auf semantischer Ebene schließlich treten Probleme auf, wenn Ausdrücke und Funktionen verlangt werden, die das System nicht kennt oder unterstützt (engl. *out-of-domain*). Hinzu

kommen Fehler des Benutzers, die – wie auch von Standard-Software durch den *Undo* Knopf bekannt – wieder rückgängig gemacht werden müssen.

Die Aufgabe von Algorithmen zur Korrektur von Mißverständnissen ist es, bei Auftreten eines der drei angegebenen Fälle dem Benutzer die Gelegenheit zu geben, im Diskurs etablierte nicht intendierte Information zu widerrufen. Dies teilt sich auf in zwei Unteraufgaben, nämlich fehlerhafte Information durch sprachliche Äußerungen zu identifizieren und aus dem Diskurs zu entfernen, und die intendierte Information durch sprachliche Äußerungen zu vermitteln. Die obige Analyse zeigt jedoch, warum dies schwierig ist: Algorithmen zur Behebung von Mißverständnissen müssen *in das System eingebaute Beschränkungen ausgleichen*, oder informell ausgedrückt: "wenn's beim erstenmal schiefgeht, warum sollte es beim zweitenmal richtig funktionieren?" Dazu kommen andere Faktoren, die bei Mißverständnissen auftreten, wie beispielsweise Hyperartikulation [Soltau and Waibel, 2000], die zusätzlich die Spracherkennung erschweren.

In diesem Kapitel wird deswegen ein Interaktionsmuster vorgestellt, das, in fünf Varianten instantiiert, die Korrektur von Fehlinterpretationen implementiert. Die Varianten erlauben entweder, den Dialog neu zu starten, die letzte Äußerung zurückzunehmen, einen Teil der letzten Äußerung zurückzunehmen oder einen Teil der letzten Äußerung durch etwas anderes zu ersetzen. Diese fünf Fälle werden in den folgenden Abschnitten näher erläutert.

9.1 Einleitung

Von der Anwendungsseite her betrachtet, impliziert das Zurücknehmen von Information das Invertieren von Seiteneffekten der im Diskurs etablierten Information. Dies bedeutet, daß Dienstanforderungen, die von zurückzunehmender Information unterstützt werden, invertiert werden müssen. Deswegen sind Korrektur-Interaktionsmuster dadurch ausgezeichnet, daß $\mathbf{F}' \neq \emptyset$ gilt. Dies ist im Gegensatz zum im letzten Kapitel vorgestellten Fragen-Interaktionsmuster. Dort galt unter Annahme der Monotonie $\mathbf{F}' = \emptyset$.

Ein weiterer Unterschied zu dem Frage-Interaktionsmuster ist, daß das Korrektur-Interaktionsmuster nur vom Benutzer initiiert werden kann. Im folgenden Abschnitt wird diskutiert, warum dies der Fall ist.

9.2 Falluntersuchung

Wie oben ausgeführt, gibt es verschiedene Gründe, in denen eine Fehlinterpretation wieder rückgängig gemacht werden muß. Alle beschriebenen Situationen sind durch eine Diskrepanz zwischen der Intention des Benutzers und der generierten (partiellen) Repräsentation dieser Intention gekennzeichnet. Diese Situationen erfordern alle Benutzer-initiierte Korrektur-Mechanismen. Die Annahme, daß es lediglich Benutzer-initiierte Instantiierungen von Korrektur-Interaktionsmustern gibt, impliziert, daß das Dialogsystem niemals in eine Lage gebracht wird, in der eine system-initiierte Instantiierung eines Interaktionsmusters erforderlich ist.

Damit stellt sich die Frage, in welchen Situationen *system-initiierte* Instantiierungen des Korrekturmusters erforderlich sind. Das Problem ist hier zunächst, für das Dialogsystem festzustellen, daß eine Korrektur vorgenommen werden muß. Bedingt durch die geringe Ausdrucksstärke der den Merkmalsstrukturen zugrundeliegenden Beschreibungssprache, erlauben die unterstützten Inferenzprozeduren nicht, sehr aussagekräftige Schlüsse zu ziehen. Deshalb wurde sich in der vorliegenden Implementierung darauf beschränkt, lediglich Typ- und Konsistenzüberprüfungen vorzunehmen.

Hier wird dann eine wesentliche vereinfachende Annahme gemacht, nämlich, daß die Überprüfung bereits vor der Inkorporation der semantischen Repräsentation in den Diskurs vorgenommen werden kann. Mit anderen Worten, es werden nur konsistente und wohltypisierte Merkmalsstrukturen in den Diskurs aufgenommen. Daraus folgt, daß die dem System mitgegebenen Inferenzmechanismen niemals die Notwendigkeit einer systeminitiierten Instantiierung des Korrekturmusters erkennen können. Somit entfällt die Notwendigkeit deren Implementierung.

Auf der anderen Seite muß das System Informationen aus dem Diskurs zurücknehmen, wenn die Annahmen, unter denen Information im Diskurs etabliert worden ist, sich im Laufe der Zeit ändern. Dies kann zum Beispiel dann passieren, wenn die Referenzen für definite Beschreibungen ungültig werden. In einem Flugbuchungssystem beispielsweise kann ein Flugticket von konkurrierenden Systemen reserviert werden bevor der Dialog zur Flugbuchung zuende geführt werden konnte. Das Dialogsystem mu dann in der Lage sein, die das Flugticket repräsentierende Information aus dem Diskurs zu entfernen und den Benutzer davon Mitteilung zu machen. Dies wäre ein Beispiel von systeminitiierter Korrektur. Allerdings wurde dieser Fall bei der Implementierung des vorliegenden Systems außer Acht gelassen.

Folglich sind die abstrakten Dialogzustände, in denen ein Korrektur-Interaktionsmuster instantiiert werden muß, gegeben durch diejenigen Zustände, in denen der Sprechakt des Benutzers die Anforderung des Korrektur-Interaktionsmusters anzeigt. Gegeben den in Definition 3 beschriebenen Sprechakt-Katalog, stellen sich die Dialogzustände, in denen die Instantiierung des Korrektur-Interaktionsmuster notwendig ist, wie folgt dar.

Jede Instantiierung des Interaktionsmusters entfernt Information aus dem Diskurs und fügt möglicherweise neue Information hinzu. Welche Information entfernt und hinzugefügt werden soll, muß auf von der Instantiierung abhängige Weise bestimmt werden. Im Falle der NEUSTART- und WIDERRUFEN-Instantiierungen ist die zu entfernende Information indirekt durch den Diskurs gegeben; die NEUSTART-Instantiierung entfernt die Information des gesamten Diskurses, während die WIDERRUFEN-Instantiierung lediglich die letzte Äußerung widerruft. Dies hat den Vorteil von robuster Ausführung der Interaktionsmuster selbst bei schlechter Spracherkennung. Im Gegensatz hierzu kann bei den verbleibenden drei Interaktionsmuster die hinzuzufügende (im Falle der Überschreiben-Instantiierung), die zu entfernende (im Falle der PARTIELLES WIDERRUFEN-Instantiierung) oder gleichzeitig die hinzuzufügende und die zu entfernende Information (im Falle der KORREKTUR-Instantiierung) explizit angegeben werden. Dies ist in Tabelle 9.2 zusammengefaßt.

Systemzustand	**Instantiierung**	**Zweck**
$\langle \_, \_, sa\_startover, \_, \_, \_ \rangle$	NEUSTART	Neustart des Dialogs
$\langle \_, \_, sa\_scratchthat, \_, \_, \_ \rangle$	WIDERRUFEN	Widerrufen der letzten Äußerung
$\langle \_, \_, sa\_overwrite, \_, \_, \_ \rangle$	ÜBERSCHREIBEN	Überschreiben einer implizit gegebenen Information durch explizit gegebene Information
$\langle \_, \_, sa\_retract, \_, \_, \_ \rangle$	PARTIELLES WIDERRUFEN	Entfernen explizit gegebener Information
$\langle \_, \_, sa\_repair, \_, \_, \_ \rangle$	KORREKTUR	Partielles Ersetzen der Information

Tabelle 9.1. Abstrakte Dialogzustände, in denen die Instantiierung des Korrektur-Interaktionsmusters notwendig ist.

Instantiierung	**zu entfernende Information**	**hinzuzufügende Information**
NEUSTART	indirekt: gesamter Diskurs	—
WIDERRUFEN	indirekt: letzte Äußerung	—
ÜBERSCHREIBEN	implizit: die mit der hinzuzufügenden Information inkompatible Information	explizit gegeben
PARTIELLES WIDERRUFEN	explizit gegeben	—
KORREKTUR	explizit gegeben	explizit gegeben

Tabelle 9.2. Bestimmung der durch die Instantiierung betroffenen Information

Ein Sonderfall ist die PARTIELLES ÜBERSCHREIBEN Instantiierung des Interaktionsmusters. Dieser Fall tritt auf, wenn der Benutzer nur die korrekte Information wiederholt, aber die fehlerkannte Information nicht explizit angibt. Dies ist beispielsweise in folgendem Dialog der Fall:
"I need directions to an Italian Restaurant"
"Would you like to go to a Chinese, Japanese or Thai place?"
"No I said Italian"
In diesem Beispiel muß das System selbst inferieren, daß die augenscheinlich als *Asian* verstandene Information aus dem Diskurs entfernt werden muß. Dies geschieht mithilfe der in Kapitel 3 beschriebenen partiell unifizierten Merkmalsstrukturen.

9.3 Das Korrektur-Interaktionsmuster

Den obigen Überlegungen zufolge, ergibt sich folgende Definition des Korrektur-Interaktionsmusters.

Definition 1 (Korrektur-Interaktionsmuster). *Ein* Korrektur-Interaktionsmuster *ist ein Interaktionsmuster* $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}' \rangle$, *wobei* $I = user$, $SA_I \subseteq \{sa\_startover, sa\_scratchthat, sa\_retract, sa\_overwrite, sa\_correct\}$, $SA_O = \{sa\_inform\}$, $\mathbf{F}$ *ist die möglicherweise leere Menge von Repräsentationen der zu widerrufenden Information, und* $\mathbf{F}' \neq \emptyset$ *enthält die zurückzunehmende Information.*

Wie aus der Definition ersichtlich ist, enkapsuliert das Korrektur-Interaktionsmuster Mechanismen, die die Zurücknahme von Informationen aus dem Diskurs und das Widerrufen von angeforderten Diensten ermöglichen. Tabelle 9.1 stellt notwendige Bedingungen zur Instantiierung eines Korrektur-Interaktionsmusters dar.

In den folgenden Abschnitten werden die fünf verschiedenen Instantiierungen des Korrektur-Interaktionsmusters beschrieben.

9.3.1 NEUSTART Instantiierung

Es ist gelegentlich sinnvoll, einen ganzen Dialog zu wiederholen. Das NEUSTART Interaktionsmuster implementiert dieses Verhalten.

Definition 2. NEUSTART-*Instantiierung Eine* Neustart-*Instantiierung ist eine Instantiierung des Korrektur-Interaktionsmuster mit* $SA_I = \{sa\_startover\}$.

Die NEUSTART-Instantiierung erfüllt offensichtlich die Bedingungen eines Korrektur-Interaktionsmusters. Abbildung 9.1 zeigt eine Instantiierung des NEUSTART Interaktionsmusters.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a restaurant in Squirrel Hill	
System:	act_inform	There are 19 places in Squirrel Hill.	
	act_request	Would you like to go to an Asian, American, Greek or Italian place?	
User:	act_inform	Italian	
System:	act_inform	There are five Asian places in Squirrel Hill.	
	act_request	Would you like to go to Chinese, Thai or Japanese place?	
User:	act_undo	Start over.	StartOver IS
System:	act_inform	Okay. Let' start over. How can I help you?	

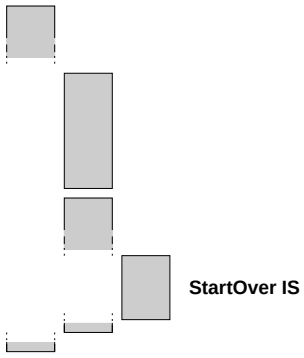


Abbildung 9.1. Ein Beispiel für die NeuStart-Instantiierung

9.3.2 Widerrufung Instantiierung

Das Widerrufung Interaktionsmuster erlaubt eine etwas genauere Zurücknahme von Information. Hier werden im Gegensatz zur Neustart Instantiierung lediglich die durch die letzte Äußerung des Initiierers etablierte Information zurückgenommen.

Definition 3. Widerrufen-*Instantiierung*
Eine Widerrufen-*Instantiierung ist eine Instantiierung des Korrektur-Interaktionsmuster mit* $SA_I = \{sa\_scratchthat\}$.

Die Widerrufen-Instantiierung erfüllt offensichtlich die Bedingungen eines Korrektur-Interaktionsmusters. Abbildung 9.2 zeigt eine Instantiierung des Widerrufen Interaktionsmusters.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a restaurant in Squirrel Hill	
System:	act_inform	There are 19 places in Squirrel Hill.	
	act_request	Would you like to go to an Asian, American, Greek or Italian place?	
User:	act_inform	Italian	
System:	act_inform	There are five Asian places in Squirrel Hill.	
	act_request	Would you like to go to Chinese, Thai or Japanese place?	
User:	act_undo	Undo.	Undo IS
System:	act_inform	Okay. Let's try this once again.	
	act_inform	There are 19 places in Squirrel Hill.	
	act_request	Would you like to go to an Asian, American, Greek or Italian place?	
User:	act_inform	Italian	
System:	act_inform	I am giving you directions to 'Sweet Basil' in Squirrel Hill.	

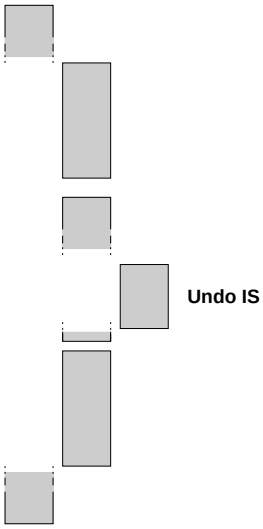


Abbildung 9.2. Ein Beispiel für die Widerrufen-Instantiierung

9.3.3 Partielle Widerrufung **Instantiierung**

Die Instantiierung eines Korrektur-Interaktionsmusters durch eine partielle Widerrufung ermöglicht eine noch feinere Unterscheidung der Information, die zurückgenommen werden soll. Im Gegensatz zur Widerrufungs Instantiierung muß die zurückzunehmende Information nicht notwendigerweise durch die letzte Äußerung in den Diskurs eingebracht werden, noch muß eine ganze Äußerung zurückgenommen werden.

Definition 4. Partielle Widerrufen-*Instantiierung*
Ein Partielle-Widerrufungs-*Instantiierung ist eine Instantiierung des Korrektur-Interaktionsmuster mit* $SA_I = \{sa\_retract\}$.

Die beiden obigen Instantiierungen des Korrektur-Interaktionsmusters haben im Gegensatz zu der Partielle Widerrufung Instantiierung die Eigenschaft, daß die zu widerrufende Information implizit durch das Interaktionsmusters gegeben ist. Im Falle der Widerrufen-Instantiierung wird der Effekt der letzten Äußerung widerrufen, während im Falle der NeuStart-Instantiierung die Effekte aller stattgefundenen Sprechakte widerrufen werden. Im Gegensatz dazu muß bei einer Partiellen Widerrufen-Instantiierung die zu widerrufende Information explizit repräsentiert (und damit auch vom Spracherkenner erkannt und von der Satzanalyse korrekt analysiert) werden. Dies hat natürlich Auswirkungen auf die Performanz der Interaktionsmuster. Die Fehlerrate einer Äußerung, die eine partielle Widerrufung beschreibt, muß notwendigerweise höher sein als die, die einen Neustart oder eine Widerrufung beschreibt.

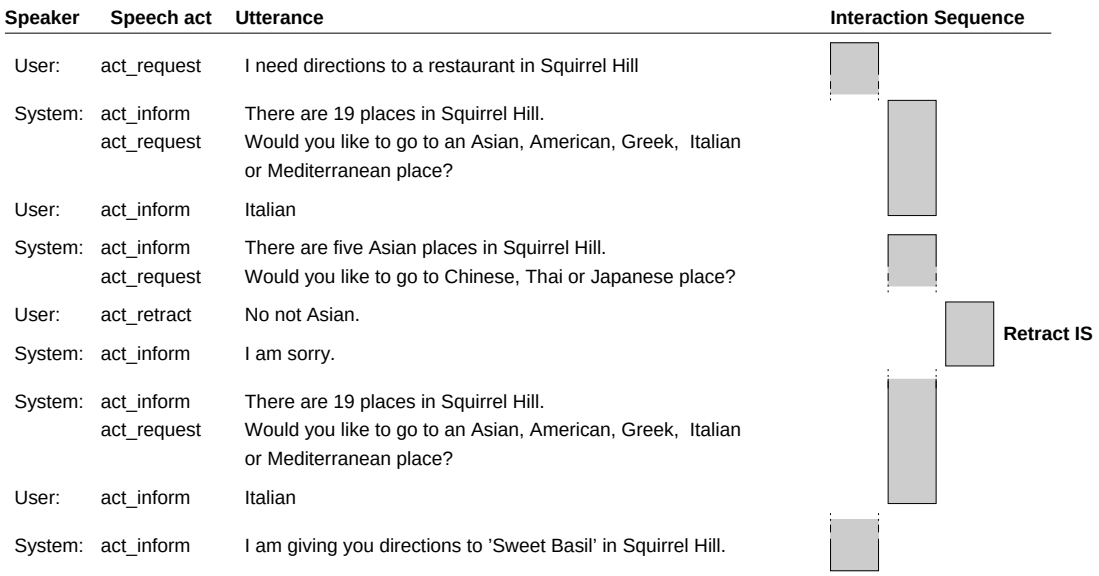


Abbildung 9.3. Ein Beispiel für die Partielle Widerrufen-Instantiierung

9.3.4 Überschreiben **Instantiierung**

Die Überschreiben Instantiierung ist das Gegenstück zu der Partiellen Widerrufung Instantiierung. Während bei obiger Instantiierung die zu widerrufende Information angegeben wird, wird bei der Überschreiben Instantiierung die korrekte Information angegeben. Das Dialogsystem muß dann eigenständig die zu widerrufende Information aus dem Diskurs bestimmen. Damit ergibt sich folgende Definition.

Definition 5. Überschreiben-*Instantiierung*
Eine Überschreiben-*Instantiierung ist eine Instantiierung des Korrektur-Interaktionsmuster mit* $SA_I = \{sa\_overwrite\}$.

Abbildung 9.4 zeigt ein Beispiel für eine Instantiierung des Korrektur-Interaktionsmusters.

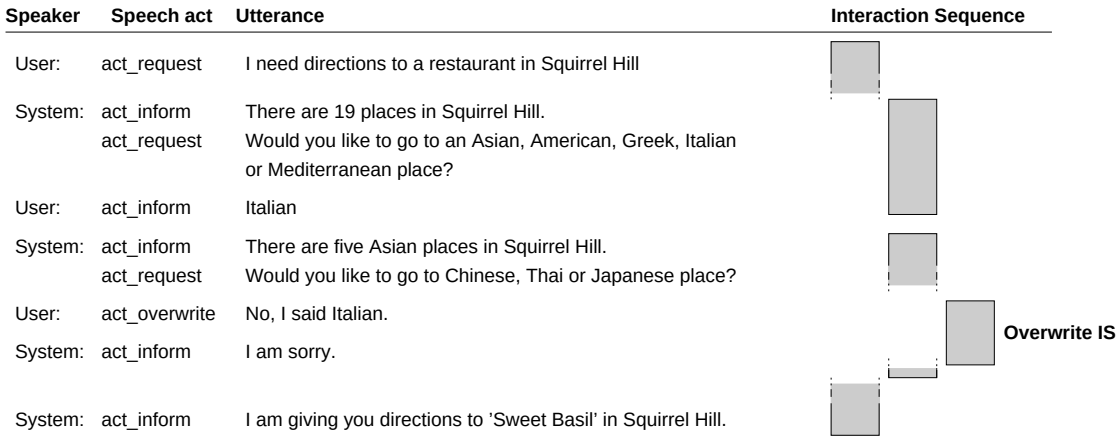


Abbildung 9.4. Ein Beispiel für eine Überschreiben Instantiierung des Korrektur-Interaktionsmusters

9.3.5 Korrektur **Instantiierung**

Die Vollständige Korrektur Instantiierung des Korrektur-Interaktionsmusters spezifiziert gleichzeitig die zurückzunehmende und die einzufügende Information. Es ist damit die Instantiierung des Korrektur-Interaktionsmusters, die die meiste Kontrolle über die zu verändernde Information erlaubt. Gleichzeitig ist es auch die Instantiierung, die die höchsten Anforderungen an die Spracherkenner- und Satzanalysekomponenten stellt.

Im Gegensatz zu den NeuStart und Widerrufen Instantiierungen muß der zurückzunehmende und der zu ersetzende semantische Inhalt explizit erwähnt werden.

Definition 6. KORREKTUR-*Instantiierung*
Eine Korrektur-*Instantiierung ist eine Instantiierung des Korrektur-Interaktionsmuster mit* $SA_I = \{sa\_correct\}$.

An dieser Stelle wird auch der Grund für die Entscheidung in Abschnitt 6.4.2 deutlich, den Sprechakt *sa_correct* von *sa_retract* und *sa_overwrite* erben zu lassen: die Funktionalität des Interaktionsmusters ist die Summe der PARTIELLE WIDERRUFUNG und ÜBERSCHREIBEN Interaktionsmuster. Abbildung 9.5 zeigt ein Beispiel für eine VOLLSTÄNDIGE KORREKTUR Instantiierung.

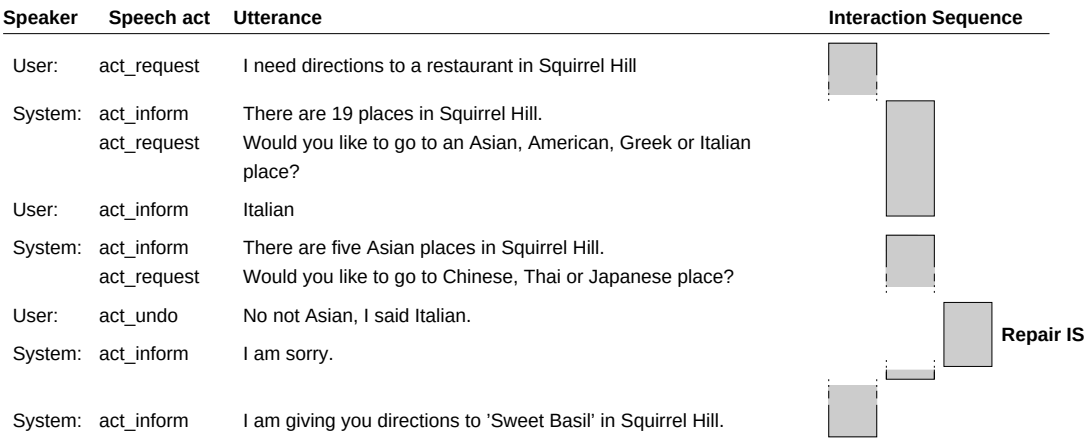


Abbildung 9.5. Ein Beispiel für eine KORREKTUR-Instantiierung

9.4 Implementierung des Korrektur-Interaktionsmusters

9.4.1 Bestimmen der zu invertierenden Dienstanforderungen

Die Definition des Korrektur-Interaktionsmusters zeichnet sich dadurch aus, daß $\mathbf{F}' \neq \perp$; in anderen Worten, Information muß aus dem Diskurs entfernt werden. Unterstützt die zu entfernende Information jedoch einen Dienstanforderungen, muß der Seiteneffekt des Dienstanforderungen ebenfalls invertiert werden. Zur Erinnerung sei erwähnt, daß Dienstanforderungen durch drei verschiedene Mechanismen abgesetzt werden können, nämlich durch objekt-orientierte Merkmalsstrukturen (siehe Abschnitt 2.5.1), Datenbankanfragen (siehe Abschnitt 5.3) und Dialogzielbeschreibungen (siehe Abschnitt 5.4). Es kann vorkommen, daß Zwischenabhängigkeiten zwischen den Diensten bestehen, die das Bestimmen der zu invertierenden Dienstanforderungen komplizierter macht.

Als Beispiel sei ein System gegeben, welches Sonderangebote für Vielflieger anbietet. Die Dialogzielbeschreibungen erlauben es, entweder den Vielfliegerstatus abzurufen (Silber, Gold, Platin) oder sich die Sonderangebote anzeigen zu lassen. Die Sonderangebote sind abhängig vom Vielfliegerstatus. Die Funktionalität des Systems ist durch zwei Anwendungsdienste implementiert, nämlich

```
   getFrequentFlyerStatus ( in : FrequentFlyerNumber,
out: FrequentFlyerStatus); displayOffers ( in
:FrequentFlyerStatus);
```

Offensichtlich hängt die Dienstanforderung, die Sonderangebote vorlesen zu lassen, nicht direkt von der Dienstanforderung, den Vielfliegerstatus zu bestimmen, ab. Ist jedoch die Vielfliegernummer fehlerkannt worden und wird durch die Instantiierung eines Korrektur-Interaktionsmusters verbessert, muß die Anforderung zum Anzeigen der Sonderangebote ebenfalls widerrufen werden, da diese Anforderung nur aufgrund des Ergebnisses der ersten Dienstanforderung gemacht worden konnte.

9.4.2 Algorithmische Bestimmung der zu widerrufenden Information

Tabelle 9.1 zeigt einen Fall, für den ein Parameter des Korrektur-Interaktionsmusters algorithmisch bestimmt werden muß. Die hier gemachte Annahme ist, daß zum einen die zu widerrufende Information in der letzten Äußerung im Diskurs etabliert worden ist und zweitens die zu widerrufende Information semantisch ähnlich zu der korrekten Information ist. Die erste Annahme kann gemacht werden, weil bei einer späteren Korrektur nicht sicher inferiert werden kann, welche Information zu widerrufen ist. Die zweite Annahme kann deswegen gemacht werden, weil der Benutzer annimmt, daß lediglich das Erwähnen der korrekten Information dem Kommunikationspartner genügend Information vermittelt, um die zu widerrufende Information selbst zu bestimmen. Das heißt, es muß "offensichtlich" sein, was die zu widerrufende Information ist, wenn in der letzten Äußerung mehrere Informationen übermittelt werden.

9.4.3 Seiteneffekte auf den Dialogzustand

Die Tatsache, daß der Benutzer eine Korrektur der Information im Diskurs anfordert, zeigt, daß im Dialog Fehler aufgetreten sind. Diese Tatsache soll im abstrakten Dialogzustand vermerkt werden, so daß eine Qualitätskontrolle des fortschreitenden Dialogs vorgenommen werden kann. Das heißt, daß sich der Wert der Variable OverallQuality bei Instantiierung des Korrektur-Interaktionsmusters verringert.

Desweiteren müssen natürlich die Werte für die Variablen Referenz und Intention nach Zurücknahme von Information neu berechnet werden.

9.5 Zusammenfassung

In diesem Kapitel wurde das Korrektur-Interaktionsmuster eingeführt. Das Korrektur-Interaktionsmuster erlaubt es, Informationen aus dem Diskurs zu entfernen und gegebenenfalls durch neue Information zu ersetzen. Es wurden fünf verschiedene Instantiierungen des Korrektur-Interaktionsmusters beschrieben. Es wurde gezeigt, daß die unterschiedliche Mächtigkeit der Instantiierungen invers proportional zu den Anforderungen an die Spracherkenner- und

Satzanalysekomponenten ist. Damit ermöglicht das Dialogsystem dem Benutzer, abhängig von der Qualität des Dialogs die geeignete Instantiierung des Korrektur-Interaktionsmusters vorzunehmen.

Dies enthebt das Dialogsystem (oder die Umgebung des Dialogsystems, wie zum Beispiel Hinweisschilder neben dem Bildschirm eines Kiosksystems) natürlich nicht der Verantwortung, dem Benutzer die Möglichkeiten dieser Instantiierungen des Interaktionsmusters mitzuteilen. Der Benutzer wird diese Instantiierungen nur dann richtig anwenden können, wenn er um ihre Eigenschaften weiß. Von einem vollständig natürlichen Korrektur-Algorithmus ist die vorliegende Implementierung noch entfernt.

Kapitel 10

Anpassen der Repräsentationen

In graphischen Benutzerschnittstellen wird dem Anwender die Möglichkeit gegeben, mittels Menüs oder Auswahlknöpfen eine von mehreren Optionen auszuwählen. Die graphische Darstellung informiert den Anwender über die Optionen zu jedem Zeitpunkt des Mensch-Maschine-Dialogs. Bei natürlichsprachlichen Dialogsystemen stellt sich die Situation anders dar. In vielen Fällen gibt es keine Möglichkeit, dem Anwender visuell Auswahlmöglichkeiten zu präsentieren. Es kann von daher vorkommen, daß der Anwender auf nichtexistente Objekte referiert. Das folgende Beispiel illustriert diesen Punkt.

Anwender: *"Give me directions to a French Restaurant in Squirrel Hill."*
System: *"I do not know a French restaurant in Squirrel Hill but I do know French places in Shadyside, South Side and East Liberty. Which neighborhood would you like to go to?*
Anwender: *"Shadyside"*
System: *"I am giving you directions to 'Le Perroquet'."*

In diesem Beispiel teilt das Dialogsystem dem Anwender mit, daß das gewünschte Objekt nicht existiert. Darüber hinaus bietet es Auswahlmöglichkeiten an, von denen angenommen werden kann, daß sie dem Anwender weiterhelfen.

Ein Interaktionsmuster, daß solche Interaktionen unterstützt, erhöht somit den Grad der Kooperation des Dialogsystems.

10.1 Einleitung

In den vorhergehenden zwei Kapitel wurden Interaktionsmuster beschrieben, die es erlauben, Information dem Diskurs hinzuzufügen und zu widerrufen. Die beiden Interaktionsmuster ermöglichen es allerdings nicht, auf einfache Weise überspezifizierte Information zu widerrufen. Diese Situationen entstehen, weil der Benutzer entweder auf Objekte referiert, die mit den gewünschten Eigenschaften so nicht existieren, oder die Fähigkeiten des Systems falsch eingeschätzt werden. Im obigen Beispiel gibt es kein Objekt, das den Anforderungen des Benutzers genügt. Das Dialogsystem initiiert daher ein Interaktionsmuster, mithil-

fe dessen die Constraints abgeschwächt werden können. In diesem Kapitel wird ein Interaktionsmuster eingeführt, das diesen Mechanismus implementiert.

Es können verschiedene Situationen auftreten, in denen das Anpassen von Repräsentationen an die tatsächlichen Gegebenheiten erforderlich ist. Das obige Beispiel illustriert den Fall, in dem es kein Objekt in der Domäne gibt, das den Anforderungen des Benutzers genügt. Dieselbe Situation tritt jedoch auch auf, wenn – durch Mißverständnis des Benutzers oder Fehlerkennung des Spracherkenners – versucht wird, ein "semantisch unsinniges" Dialogziel zu erreichen. Als Beispiel dient der folgende Dialog:

Anwender: *"Give me directions to a Giant Eagle"*
Erkennung: *"Give me directions to current location."*
System: *"You can either ask for directions to some place or inquire about your current location. If you want to get directions, please say: 'give directions'. If you want to know your current location, please say 'location'."*
Anwender: *"give directions."*
System: *"Where would you like to go to?"*

In diesem Beispiel wird das Wissen ausgenutzt, daß man nicht Fahrtanweisungen zu seinem gegenwärtigen Standpunkt anfragen kann. Dies ist im Aufgabenmodell kodiert. Die beiden involvierten Dialogzielbeschreibungen, nämlich die für die Fahrtanweisungen und das für das Mitteilen des gegenwärtigen Standpunktes, verlangen entsprechende Informationen.

Eine ähnliche Situation kann auch bei Datenbankwächtern auftreten. Datenbankwächter sind, genau wie Dialogzielbeschreibungen, durch Merkmalsstrukturen repräsentiert. Nur partiell passende Repräsentationen im Diskurs geben genauso Anlaß zu einem Anpassungsdialog im Falle der Datenbankwächter wie im Falle der Dialogzielbeschreibungen.

Ein letzter Fall schließlich, in dem das Anpassen von Benutzerangaben erforderlich ist, ist gegeben durch nicht wohl-typisierbare Repräsentationen. Die Wohlgeformtheitsbeschränkungen der typisierten Merkmalsstrukturen schränken die Kombination von Typen und Merkmalen ein. Wird von der Satzanalyse eine Repräsentation erzeugt, die die Wohlgeformtheitsbeschränkungen verletzt, kann dies entweder an einer Fehlerkennung liegen oder aber an mangelndem Wissen des Anwenders. In beiden Fällen muß das Dialogsystem dem Anwender die Möglichkeit geben, im Diskurs etablierte Information zurückzunehmen.

10.2 Falluntersuchung

Wie oben ausgeführt, dient das Interaktionsmuster dazu, den Benutzer aus einer Menge "ähnlicher" Objekte auswählen zu lassen, wenn die vom Benutzer spezifizierten Einschränkungen an das Objekt nicht erfüllbar sind. Natürlich kann dieses Interaktionsmuster in erster Linie bei fehlgeschlagenen Datenbankanfragen angewandt werden. Dies wird im abstrakten Dialogzustand dadurch erfaßt, daß der Wert für die Variable ReferierendeAusdrücke auf *inkonsistent* gesetzt wird.

Allerdings läßt es der allgemeine Entwurf des Interaktionsmusters ebenfalls zu, in anderen Situationen von überspezifizierten Constraints den Benutzer auswählen zu lassen. Dies kann beispielsweise der Fall sein, wenn sich alle Dialogzielbeschreibungen im Zustand *Inkonsistent* befinden. Diese Situation tritt auf, wenn die im Diskurs vorhandene Information inkompatibel mit allen Dialogzielbeschreibungen ist. Mit anderen Worten, das Dialogsystem sieht keine Möglichkeit, die vom Anwender angeforderten Dienste zu erfüllen.[1]

Schließlich kann diese Situation noch auftreten, wenn eine Datenbankanfrage stattfinden müßte, aber alle Datenbankwächter inkompatibel mit der vorhandenen Information sind. Die abstrakten Dialogzustände, in denen dieses Interaktionsmuster angewandt werden kann, sind in Tabelle 10.1 zusammengefaßt.

Systemzustand	**Instantiierung**	**Zweck**
$\langle \_, \_, \_, \mathit{inkonsistent}, \_, \_ \rangle$	Wächter	Repr. der Äußerung mit Datenbankwächtern kompatibel zu machen
$\langle \_, \_, \_, \_, \mathit{inkonsistent}, \_ \rangle$	Objekte	Repr. der Äußerung mit Objekten in der Datenbank kompatibel zu machen
$\langle \_, \_, \_, \_, \_, \mathit{inkonsistent} \rangle$	Dialogzielbeschreibungen	Repr. der Äußerung mit Dialogzielbeschreibungen kompatibel zu machen

Tabelle 10.1. Abstrakte Dialogzustände, in denen eine Instantiierung des ANPASSEN Interaktionsmuster vorgenommen werden kann

Wie aus Tabelle 10.1 ersichtlich, kann das Interaktionsmuster dann instantiiert werden, wenn die Repräsentationen im Diskurs inkonsistent mit den Datenbankwächtern, den Einträgen in der Datenbank oder den Dialogzielspezifikationen ist. Der Zweck ist somit, dem Benutzer die Möglichkeit zu geben, einige der von ihm spezifizierten Constraints zurückzunehmen.

10.3 Das ANPASSEN Interaktionsmuster

Die Überlegungen im vorigen Abschnitt zeigen, daß das ANPASSEN Interaktionsmuster instantiiert wird, wenn das Dialogsystem keine Möglichkeit sieht, einen Dienst der Anwendung anzufordern, der konsistent mit den Spezifikationen des Benutzers ist. Daraus folgt, daß das ANPASSEN Interaktionsmuster notwendigerweise vom Dialogsystem initiiert werden muß.

Damit kann nun das Interaktionsmuster ANPASSEN formell definiert werden.

Definition 1 (Interaktionsmuster ANPASSEN). *Das* ANPASSEN Interactionsmuster *ist ein system-initiiertes Interaktionsmuster, daß es dem Benutzer*

[1] Natürlich hat das Dialogsystem keine Möglichkeit festzustellen, ob diese Situation durch falsche Spracherkennung oder durch ein Mißverständnis des Anwenders aufgetreten ist.

erlaubt, überspezifizierte Constraints interaktiv zurückzunehmen. Es wird beschrieben durch ein Tupel $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}', \rangle$*, wobei* $I = system$,$SA_I = \{act\_inform, act\_question\}$*, und* $SA_O = \{act\_answer\}$.

Ähnlich wie beim FRAGEN-Interaktionsmuster ergibt sich die Notwendigkeit, zusätzliche Information vom Benutzer zu erfragen. Aus diesem Grunde gibt es auch beim ANPASSEN-Interaktionsmuster verschiedene Instantiierungen, die die Größe der Ergebnismenge und Konfidenzannotationen in Betracht ziehen.

10.3.1 Instantiierung mittels Disjunktionsfragen

Abbildung 10.1 zeigt eine Instantiierung des ANPASSEN-Interaktionsmusters mit einer Disjunktivfrage.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a French restaurant in Squirrel Hill	
System:	act_inform	I do not know a French place in Squirrel Hill, but I do know French Places in Shadyside, South Side and East Liberty.	Match IS
	act_request	Which neighborhood would you like to go to?	
User:	act_inform	East Liberty.	
System:	act_inform	I am giving you directions to 'La Foret' in East Liberty.	

Abbildung 10.1. Ein Beispiel für eine Instantiierung des ANPASSEN-Interaktionsmusters mit Disjunktivfragen

10.3.2 Instantiierung mittels Ersetzungsfragen

Abbildung 10.1 zeigt eine Instantiierung des ANPASSEN-Interaktionsmusters mit einer Ersetzungsfrage.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a Chinese restaurant in Fox Chapel.	
System:	act_inform	I do not know a Chinese place in Squirrel Hill, but I do know Chinese Places in other neighborhoods.	Match IS
	act_request	Which neighborhood would you like to go to?	
User:	act_inform	Shadyside.	
System:	act_inform	I am giving you directions Tai Pei' in Shadyside.	

Abbildung 10.2. Ein Beispiel für eine Instantiierung des ANPASSEN-Interaktionsmusters mit Ersetzungsfragen

10.4 Implementierung

Von den Verarbeitungsmechanismen her betrachtet, müssen zwei verschiedene Situationen unterschieden werden. Im Falle von Datenbankzugriffen, die leere Ergebnismengen liefern (die Situation entspricht der zweiten Zeile in Tabelle 10.1), müssen die Constraints der Datenbankanfrage abgeschwächt werden. In Fällen, in denen die Information im Diskurs inkonsistent mit entweder mit den Datenbankwächtern oder den Dialogzielbeschreibungen ist, müssen Constraints aus inkonsistenten Merkmalsstrukturen entfernt werden. Diese beiden Fällen werden im folgenden beschrieben.

10.4.1 Abschwächen von Datenbank-Constraints

Wie in Definition 2 beschrieben ist, können bei den Datenbankkonvertierungsregeln Relaxations-Level abgegeben werden. Damit kann beschrieben werden, welche Information im Falle leerer Ergebnismengen ausgelassen kann. Von dieser Möglichkeit wird hier Gebrauch gemacht. Es werden solange Constraints in der Datenbankanfrage ausgelassen, bis entweder kein Datenbankwächter mehr erfüllt ist (falls vorhanden) oder aber eine nichtleere Ergebnismenge zurückgeliefert wird.

10.4.2 Partiell Unifizierte Merkmalsstrukturen

In den Fällen, in denen die Information im Diskurs inkonsistent mit Datenbankwächtern (oder Dialogzielbeschreibungen) ist, muß Information wieder aus dem Diskurs entfernt werden. Um zu bestimmen, welche Constraints das Dialogsystem dem Benutzer zum Entfernen vorschlagen soll, wird die Repräsentation im Diskurs mit den Datenbankwächtern (oder den Dialogzielbeschreibungen) partiell unifiziert. Inkonsistenzen sind dann aus der partiell unifizierten Struktur abzulesen.

10.4.3 Integration mit dem Spracherkenner

Bei der Integration mit dem Spracherkenner wurde dieselbe Lösung verfolgt wie bei dem FRAGEN-Interaktionsmuster.

10.5 Diskussion

Das in diesem Kapitel vorgestellte Interaktionsmuster erlaubt es, fehlerhaft oder überspezifizierte Constraints des Benutzers abzuschwächen. Dazu werden die angegebenen Constraints sukzessive abgeschwächt, bis eine Lösung gefunden wird oder es keine abschwächbaren Constraints mehr gibt. Die $relax()$ Ausdrücke der Datenbankkonvertierungsregeln geben dabei an. welche Constraints abgeschwächt werden können und in welcher Reihenfolge. Dies ist übrigens der Grund, warum in den gezeigten Beispielen immer andere Stadtteile vorgeschlagen werden, anstatt anderer Restaurants.

Über das systematische Abschwächen von Constraints bei Datenbankzugriffen wurde bereits berichtet [Ferrieux and M.D.Sadek, 1994]. Allerdings hat

im vorliegenden Fall der Systemdesigner mehr Kontrolle darüber, in welcher Reihenfolge Constraints abgeschwächt werden sollen.

Die Initiative zur Instantiierung des ANPASSEN-Interaktionsmusters geht immer vom Dialogsystem aus. Aus diesem Grunde ist es benutzerfreundlicher als eine Instantiierung des KORREKTUR-Interaktionsmusters, da hier dem Benutzer die verfügbaren Optionen mitgeteilt werden.

10.6 Zusammenfassung

Das ANPASSEN-Interaktionsmuster erlaubt es dem Anwender des Dialogsystems, angegebene Constraints wieder zurückzunehmen. Es wurde gezeigt, wie das ANPASSEN-Interaktionsmuster – genau wie das FRAGEN-Interaktionsmuster in verschiedenen Arten instantiiert werden kann, um bestmöglich der gegebenen Situation im Diskurs Rechnung zu tragen.

Kapitel 11

Hilfe, Transfer und Kontextwechsel

Interaktionsmuster sind definiert als Interaktionen zwischen Anwender und Dialogsystem, die den Zustand des Dialogsystems "näher" an einen Zielzustand bringen. Die in den vorangegangenen drei Kapiteln beschriebenen Interaktionsmuster ermöglichen es dem Anwender, Information zum Diskurs hinzuzufügen, aus dem Diskurs wieder zu entfernen oder zu modifizieren. Das hier beschriebene Interaktionsmuster unterscheidet sich von den vorangegangenen Interaktionsmustern darin, daß es nicht zur Manipulation von Information im Diskurs dient. Die Funktion ist eher, dem Anwender (oder einer dritten Person) den Systemzustand mitzuteilen oder zu verändern, um somit Einfluß auf den ablaufenden Dialog zu nehmen.

Es gibt jedoch Fälle, in denen nicht der Dialogzustand verändert werden muß, sondern der Dialogzustand dem Anwender mitgeteilt werden muß, damit er den Dialog fortsetzen kann. Dies ist für kontextbezogene Hilfe der Fall. Aus diesem Grunde wird in diesem Kapitel ein Interaktionsmuster eingeführt, dessen Instantiierungen es erlauben, den abstrakten Dialogzustand dem Anwender mitzuteilen, oder den abstrakten Dialogzustand *direkt* zu verändern, um somit Einfluß auf den Dialogablauf zu nehmen.

11.1 Einleitung

Wir werden insbesondere vier Fällen besondere Beachtung schenken: (i) die Situation, in der der Anwender explizit Hilfe anfordert, (ii) die Situation, in der der Anwender die letzte Äußerung des Systems noch einmal höre möchte, (iii) die Situation, in der die Qualität des Dialogs soweit gesunken ist, daß eine Fortführung inpraktikabel erscheint und (iv) die Situation, in denen das Aufgabenmodell angepaßt werden muß. Die folgenden Beispiele erläutern, wie diese Situationen mit dem Dialogzustand zusammenhängen.

Im Falle der Hilfeanforderung muß dem Anwender mitgeteilt werden, "wo" er sich im Dialog befindet. Dies bedeutet, er muß darüber informiert werden, welche Dienste noch anforderbar sind, welche Information noch fehlen, um den gewünschten Dienst auszuführen, und was er machen muß, um ans Ende des

Dialogs zu kommen. [1] Diese Information läßt sich leicht aus dem abstrakten Dialogzustand generieren. Die erreichbaren Dialogziele sind repräsentiert durch die Dialogbeschreibungen, die sich im Zustand *Selektiert* oder *Eindeutig Bestimmt* befinden. Fehlende oder ambige Information kann aus den Repräsentationen im Diskurs generiert werden.

Der zuletzt geäußerte Satz des Dialogsystems kann einfach aus dem Diskursbaum gelesen und wiederholt werden.

Im Falle der abnehmenden Qualität des Dialoges ist die Situation eine ähnliche. Wird eine bestimmte Schranke unterschritten und ein Anruf beispielsweise an eine Person vermittelt, sollten die bereits im Diskurs etablierten Informationen weitergeleitet werden, um Zeit zu sparen.

Schließlich ist der Fall zu betrachten, in denen das Aufgabenmodell während des Dialogs gewechselt wird. Dies kann beispielsweise in einem interaktiven Einkaufssystem der Fall sein. Zunächst wählt der Anwender die gewünschten Waren aus, legt sie in den virtuellen Einkaufswagen, oder entfernt sie wieder. Nachdem die Einkäufe getätigt worden sind, muß die Kreditkartennummer erfragt werden. Mit anderen Worten, das Aufgabenmodell muß von den unterstützten Dialogzielbeschreibungen $addToCart()$ und $removeFromCart()$ auf $proceedToCheckout()$ wechseln. Dies kann durch explizite Anforderung des Benutzers geschehen oder durch Inferenz des Dialogsystems.

Das in diesem Kapitel vorgestellte Interaktionsmuster ermöglicht diese vier Instantiierungen.

11.2 Falluntersuchung

11.2.1 Die TRANSFER-Instantiierung

Das Dialogsystem führt Buch über die Anzahl und Länge der instantiierten Interaktionsmuster. Übersteigt die Anzahl der Klärungsfragen oder Korrekturversuche eine bestimmte Anzahl, besteht die Möglichkeit, den Dialog an eine Person zu transferieren. Dies ist beispielsweise in *Call Center* wichtig. Die bis zum Transfer erfaßte Information soll dabei an die Person weitergegeben werden.

11.2.2 Die HILFE-Instantiierung

Fragt der Anwender des Dialogsystems explizit nach Hilfe, wird das Interaktionsmuster instantiiert. Hierbei müssen dem Anwender die zur Verfügung stehenden Dialogzielbeschreibungen mitgeteilt werden sowie eine Beschreibung der Information, die er noch dem System übermitteln muß bevor eine Dialogzielbeschreibung erreicht ist.

[1] Natürlich steht dem Anwender immer die Möglichkeit offen, nach Kenntnisnahme der Hilfemeldung den Dialog abzubrechen, falls er sich an einer anderen Stelle im Dialog befindet als angenommen.

11.2.3 Die WIEDERHOLEN-Instantiierung

Bittet der Anwender des Dialogsystems explizit um die Wiederholung der letzten Äußerung, wird das Interaktionsmuster instantiiert. Hierbei muß dem Anwender einfach der zuletzt geäußerte Satz mitgeteilt werden.

11.2.4 Die KONTEXTWECHEL-Instantiierung

Die Kontextwechsel-Instantiierung kann immer dann stattfinden, wenn sich der Anwendungskontext des Systems ändert. Damit wird die Entscheidung bezüglicher dieser Instantiierung außerhalb des Dialogsystems getroffen.

11.2.5 Zusammenfassung der Falluntersuchung

Die oben beschriebenen Fälle, die die Instantiierung des ZUSTAND-Interaktionsmusters erlauben, sind in Tabelle 11.1 zusammengefaßt.

Systemzustand	**Instantiierung**	**Zweck**
$\langle \_, \_, sa\_help, \_, \_, \_ \rangle$	HILFE	dem Benutzer kontextabhängige Hilfe zu geben
$\langle \_, \_, sa\_repeat, \_, \_, \_ \rangle$	WIEDERHOLUNG	die letzte Äußerung des Dialogsystems zu wiederholen
$\langle \_, poorQuality, \_, \_, \_, \rangle$	TRANSFER	Kontrolle zu einem menschlichen Gesprächspartner transferieren
$\langle \_, \_, \_, \_, \_, \_ \rangle$	KONTEXTWECHSEL	Austausch der Wissensquellen

Tabelle 11.1. Die Klassifikation der Systemzustände, in denen das ZUSTANDS-Interaktionsmuster instantiiert werden kann.

Die Initiative der obigen Instantiierung geschieht in zwei Fällen vom Anwender, und in zwei Fällen vom System (siehe auch Tabelle 11.2. Es ist aber schwieriger, ein Interaktionsmuster so zu gestalten, daß es benutzer- und systeminitiiert werden kann. Wir werden aus diesem Grunde in diesem Kapitel zwei Interaktionsmuster einführen.

Instantiierung	**Initiator**
HILFE	Anwender
WIEDERHOLUNG	Anwender
TRANSFER	System
KONTEXTWECHSEL	System

Tabelle 11.2. Die Initiatoren des Zustands-Interaktionsmusters

11.3 Das Zustands-Interaktionsmuster

Das *Zustands*-Interaktionsmuster verändert nicht den Informationsgehalt des Diskurses. Vielmehr wird es angewandt, um den Zustand des Dialogsystems dem Anwender (im Falle der Hilfe- oder Wiederholungs-Instantiierung) oder einer dritten Person mitzuteilen (im Falle des Transfers) mitgeteilt wird oder um den Dialogzustand einer neuen Situation anzupassen (im Falle des Kontextwechsels). In allen vier Fällen muß das System dem Anwender eine entsprechende Mitteilung machen. Das ZUSTANDS-Interaktionsmuster kann demzufolge wie folgt definiert werden.

Definition 1 (ZUSTANDS-Interaktionsmuster). *Das* Anwender-initiierte ZUSTANDS-Interaktionsmuster *ist durch ein Tupel* $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}' \rangle$ *gegeben, wobei* $I = user$, $SA_I = \{sa\_inquirestate\}, SA_O = \{sa\_inform\}$, *und* $\mathbf{F} = \mathbf{F}' = \emptyset$. *Das* System-initiierte ZUSTANDS-Interaktionsmuster *ist durch ein Tupel* $\langle I, SA_I, SA_O, \mathbf{F}, \mathbf{F}' \rangle$ *gegeben, wobei* $I = system$, $SA_I = \{sa\_inform\}, SA_O = \emptyset$, *und* $\mathbf{F} = \mathbf{F}' = \emptyset$.

11.3.1 Die Wiederholen-Instantiierung

Die *Wiederholen*-Instantiierung des Interaktionsmusters ist die einfachste Instantiierung. Hier muß lediglich der zuletzt geäußerte Satz des Dialogsystems dem Anwender noch einmal präsentiert werden. Hierbei ist es auch möglich zu notieren, wie häufig Aufforderungen zur Wiederholung gemacht werden. Damit kann die Qualität des Dialogs überprüft werden.

Speaker	Speech act	Utterance
User:	act_request	I need directions to a restaurant in Squirrel Hill
System:	act_inform	There are 19 places in Squirrel Hill.
	act_request	Would you like to go to Asian, American, Italian, Greek or Mediterranean place?
User:	act_repeat	
System:	act_inform	I said:
	act_inform	There are 19 places in Squirrel Hill.
	act_request	Would you like to go to Asian, American, Italian, Greek or Mediterranean place?
User:	act_inform	Italian
System:	act_inform	I am giving you directions to 'Sweet Basil' in Squirrel Hill.

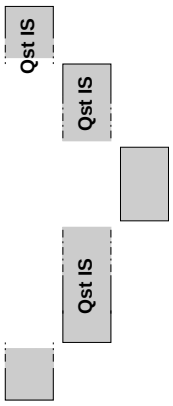


Abbildung 11.1. Ein Beispiel für eine Wiederholen-Instantiierung des ZUSTANDS-Interaktionsmusters

11.3.2 Die Hilfe-Instantiierung

Die *Hilfe*-Instantiierung des Interaktionsmusters ist muß dem Anwender die erreichbaren Dialogziele beschreiben, und, wenn nur noch ein Dialogziel erreichbar ist, die fehlende Information. Auch hier ist es auch möglich zu notieren, wie

häufig Hilfe angefordert wird. Damit kann die Qualität des Dialogs überprüft werden.

Speaker	Speech act	Utterance	Interaction Sequence
User:	act_request	I need directions to a restaurant in Squirrel Hill	
System:	act_inform	There are 19 places in Squirrel Hill.	
	act_request	Would you like to go to Asian, American, Italian, Greek or Mediterranean place?	
User:	act_help	Help	
System:	act_inform	You asked to get directions to a restaurant in Squirrel Hill.	
	act_inform	However, I need some more information from you.	
	act_request	Would you like to go to Asian, American, Italian, Greek or Mediterranean place?	
User:	act_inform	Italian	
System:	act_inform	I am giving you directions to 'Sweet Basil' in Squirrel Hill.	

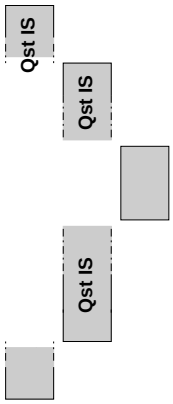


Abbildung 11.2. Ein Beispiel für eine Hilfe-Instantiierung des ZUSTANDS-Interaktionsmusters

11.3.3 Die Transfer-Instantiierung

Die Aufgaben der Transfer-Instantiierung des Interaktionsmusters sind einfach. Hier müssen lediglich bereits erfragte Informationen aus dem Diskurs gesammelt, der Anwendung entsprechend aufbereitet und an einen *Call Center*-Operator weitergeleitet werden. Dem Anwender des Systems ist eine entsprechende Mitteilung zu machen.

11.3.4 Die Kontextwechel-Instantiierung

Die Kontextwechsel-Instantiierung wird von der Anwendung initiiert. Mit anderen Worten, eine Komponente außerhalb des Dialogsystems sorgt dafür, daß die durch das Dialogsystem angebotenen Dienste gewechselt werden. Dem Anwender ist eine entsprechende Mitteilung zu machen.

11.4 Implementierung

Die Implementierung der beschriebenen Instantiierung ist einfach; in den Hilfe-, Wiederholen- und Transfer-Instantiierungen muß lediglich Information aus dem Dialogzustand erfragt und entsprechend aufbereitet werden. Die Kontextwechsel-Instantiierung erfolgt durch das Auswechseln der Dialogzielbeschreibungen im Aufgabenmodel.

11.4.1 Seiteneffekte auf den Dialogzustand

Der Dialogzustand kann im Falle der Hilfe- und Wiederholen-Instantiierungen angepaßt werden, um der abnehmenden Qualität des Dialogs Rechnung zu tragen. Hierzu kann der Wert der Variable GESAMTQUALITÄT heruntergesetzt

werden. Im Falle der systeminitiierten Instantiierungen des Interaktionsmusters wird der gegenwärtige Dialog mit dem System abgebrochen. Im Falle des Transfers findet kein weiterer Dialog mit dem System statt, im Falle des Kontextwechsels wird das System in den Anfangszustand versetzt und der Dialog kann mit dem neuen Aufgabenmodell fortgesetzt werden.

11.4.2 Integration mit dem Spracherkenner

Die Integration mit dem Spracherkenner ist ebenfalls einfach. Im Falle der Anwender-Initiierten Instantiierungen wird der Spracherkenner in denselben Zustand versetzt, in dem er vor Instantiierung des Interaktionsmusters war. Im Falle der System-Initiierten Interaktionsmuster wird der Spracherkenner in denselben Zustand versetzt, in dem er zu Beginn des Dialoges war.

11.5 Zusammenfassung

Das hier vorgestellte Interaktionsmuster ermöglicht es, den abstrakten Dialogzustand direkt mit in den Dialog einzubeziehen. Somit ist es möglich, eine kontextbezogene Hilfe direkt und einfach zu implementieren. Desweiteren ist die Möglichkeit hilfreich, jederzeit die letzte Aussage des Systems wiederholen lasen zu können. Die Transfer-Instantiierung ermöglicht es, fehlschlagende Dialoge durch Personen zu retten. Schließlich ermöglicht es die Kontextwechsel-Instantiierung, den Dialog in mehrere aufeinanderfolgende Schritte einzuteilen. Dadurch hat die Anwendung mehr Kontrolle darüber, welche Dienste zu einem gegebenen Zeitpunkt im Dialog angefordert werden können.

Kapitel 12

Ein Katalog von Interaktionsmustern

Die in den vorigen Kapiteln beschriebenen Interaktionsmuster werden in Abhängigkeit des Dialogzustandes instantiiert. In diesem Kapitel wird der Zusammenhang zwischen Dialogzuständen und Interaktionsmustern beschrieben.

12.1 Einleitung

Vom Standpunkt der Dialogverarbeitung ist der Zusammenhang zwischen abstraktem Dialogzustand und Interaktionsmuster am wichtigsten, da dieser das Verhalten des Dialogsystems bestimmt. Aus diesem Grunde wurde bei der Einführung des jeweiligen Interaktionsmusters eine Menge von Systemzuständen angegeben, in denen das Interaktionsmuster instantiiert werden kann.

Es ist allerdings auch von anderen Standpunkten her interessant, die Interaktionsmuster zu klassifizieren. Bei Unvollständigkeiten können somit fehlende Interaktionsmuster entdeckt werden. Aus diesem Grunde werden die Interaktionsmuster hier nach mehreren Kriterien klassifiziert. Weiterhin wird beschrieben, wie die Diskursstruktur mit den Interaktionsmustern interagiert. Darüber hinaus wird auch beschrieben, wie die Instantiierung der Interaktionsmuster mit den Wissensquellen interagiert.

12.2 Klassifizierungen der Interaktionsmuster

Tabelle 12.1 faßt die in den vorigen Kapiteln beschriebenen Interaktionsmuster, die Systemzustände, in denen die Interaktionsmuster angewandt werden können, und die Instantiierungen der Interaktionsmuster zusammen. Diese Tabelle ist eine Zusammenfassung der bei der Einführung des jeweiligen Interaktionsmusters angegebenen Tabellen.

Nicht alle möglichen abstrakten Dialogzustände sind in Tabelle 12.1 aufgeführt. In diesen Fällen wird kein Interaktionsmuster instantiiert. Stattdessen wird ein vordefiniertes Verhalten ausgeführt.

Die Interaktionsmuster können ebenfalls nach Art der Aktualisierung der Information im Diskurs klassifiziert werden.

Interaktionsmuster	**Systemzustand**	**Instantiierung**	**Zweck**
KLÄRUNGSFRAGEN	$\langle poorQuality, \_, \_, \_, \_\rangle$	Ja/Nein Frage	Bestätigung unsicherer Information
	$\langle \_, \_, Selected, \_, \_\rangle$	Ersetzungsfrage	Erfragen fehlender Information
		Disjunktivfrage	für Datenbankzugriff
	$\langle \_, \_, \_, \_, AmbiguousReference, \_\rangle$	Disjunktivfrage	Disambiguierung ambiger Referenzen
	$\langle \_, \_, \_, \_, \_, Selected\rangle$	Disjunktivfrage	Disambiguierung mehrerer möglicher Ziele
	$\langle \_, \_, \_, \_, \_, Determined\rangle$	Disjunktivefrage	Erfragen fehlender Information
		Ersetzungsfrage	für die gewählte Dialogzielbeschreibung
KORREKTUR	$\langle \_, \_, sa\_startover, \_, \_, \_\rangle$	NEUSTART	Neustart des Dialogs
	$\langle \_, \_, sa\_scratchthat, \_, \_, \_\rangle$	WIDERRUFEN	Widerrufen der letzten Äußerung
	$\langle \_, \_, sa\_overwrite, \_, \_, \_\rangle$	ÜBERSCHREIBEN	Überschreiben einer implizit gegebenen Information durch explizit gegebene Information
	$\langle \_, \_, sa\_retract, \_, \_, \_\rangle$	PARTIELLES WIDERRUFEN	Entfernen explizit gegebener Information
	$\langle \_, \_, sa\_repair, \_, \_, \_\rangle$	KORREKTUR	Partielles Ersetzen der Information
ANPASSEN	$\langle \_, \_, \_, inconsistent, \_, \_\rangle$	WÄCHTER	Repr. der Äußerung mit Datenbankwächtern kompatibel zu machen
	$\langle \_, \_, \_, \_, inconsistent, \_\rangle$	OBJEKTE	Repr. der Äußerung mit Objekten in der Datenbank kompatibel zu machen
	$\langle \_, \_, \_, \_, \_, inconsistent\rangle$	DIALOGZIELBESCHREIBUNGEN	Repr. der Äußerung mit Dialogzielbeschreibungen kompatibel zu machen
ZUSTAND	$\langle \_, \_, sa\_help, \_, \_, \_\rangle$	HILFE	dem Benutzer kontextabhängige Hilfe zu geben
	$\langle \_, \_, sa\_repeat, \_, \_, \_\rangle$	WIEDERHOLUNG	die letzte Äußerung des Dialogsystems zu wiederholen
	$\langle \_, poorQuality, \_, \_, \_, \rangle$	TRANSFER	Kontrolle zu einem menschlichen Gesprächspartner transferieren
	$\langle \_, \_, \_, \_, \_, \_\rangle$	KONTEXTWECHSEL	Austausch der Wissensquellen

Tabelle 12.1. Übersicht über die Interaktionsmuster, die Dialogzustände, in denen sie angewandt werden können und Instantiierungen der Interaktionsmuster

Interaktionsmuster	Instantiierung	F	F'
KLÄRUNGSFRAGEN	jede	$\neq \emptyset$	$= \emptyset$
KORREKTUR	NEUSTART	$= \emptyset$	$\neq \emptyset$
	WIDERRUFEN	$= \emptyset$	$\neq \emptyset$
	ÜBERSCHREIBEN	$\neq \emptyset$	$\neq \emptyset$
	PARTIELLES WIDERRUFEN	$= \emptyset$	$\neq \emptyset$
	KORREKTUR	$\neq \emptyset$	$\neq \emptyset$
ANPASSEN	jede	$\neq \emptyset$	$\neq \emptyset$
ZUSTAND	jede	$= \emptyset$	$= \emptyset$

Tabelle 12.2. Klassifikation der Interaktionsmuster

Eine Klassifizierung der Interaktionsmuster nach Initiator ist in Tabelle 12.3 dargestellt.

Interaktionsmuster	System-Initiiert	Benutzer-Initiiert
FRAGE	•	
KORREKTUR		•
ANPASSEN	•	
ZUSTAND	•	•

Tabelle 12.3. Eine Klassifizierung der Interaktionsmuster nach Initiator

12.3 Interaktionsmuster und die Struktur des Diskurses

Die Instantiierung von Interaktionsmustern induziert die Struktur des Diskurses. Der Algorithmus erzeugt einen Unterbaum immer dann, wenn der neue Turn nicht Element der gegenwärtigen Instantiierung ist (Operation (b) in Abbildung 5.2). Desweiteren wird der gegenwärtige Unterbaum nur dann abgeschlossen, wenn die aktuelle Instantiierung beendet ist (Operation (c) in Abbildung 5.2). Dadurch wird eine Diskursstruktur mit der Eigenschaft erzeugt, daß eine Instantiierung eines Interaktionsmusters nur eine Ebene des Diskursbaumes belegt.

Es ist folglich wichtig zu bestimmen, ob ein Turn zur aktuellen Instantiierung eines Interaktionsmusters gehört oder nicht. Ein Turn gehört zur gegenwärtigen Instantiierung eines Interaktionsmusters, wenn er alle Constraints des gegebenen Interaktionsmusters erfüllt. Mit Constraints der Interaktionsmuster sind die Bedingungen der Interaktionsmuster an die Instantiierungen gemeint, beispielsweise, daß der Initiator des FRAGEN-Interaktionsmusters den Sprechakt *sa_question* verwenden muß, oder daß ein KORREKTUR-Interaktionsmuster benutzer-initiiert sein muß. Aus dieser Anforderung ergibt sich die Notwendigkeit der Sprechakterkennung. In Abhängigkeit des Sprechaktes wird dann ein Interaktionsmuster instantiiert oder eine existierende Instantiierung fortgeführt oder beendet.

12.3.1 Sprechakterkennung

Eine Sprechakterkennung ist nur dann notwendig, wenn der Sprechakt nicht offen genannt worden ist. In manchen Fällen ist der Sprechakt direkt aus der semantischen Repräsentation der Äußerung ablesbar. Tabelle 12.4 zeigt die Liste der offen erkennbaren Sprechakte an. In den Fällen, in denen der Sprechakt nicht offen erkannt werden kann, müssen andere Wissensquellen zu Hilfe genommen werden.

Sprechakt	**offen erkennbar**	**assoziiert mit Interaktionsmuster**
sa_question	ja	Klärungsfrage
sa_whquestion		
sa_disjquestion		
sa_ynquestion		
sa_repair	ja	Korrektur
sa_startover		
sa_scratchthat		
sa_overwrite		
sa_retract		
sa_correct		
sa_command	ja	Zustand
sa_help		
sa_repeat		
sa_answer	nein, Diskurs	Klärungsfrage
sa_inform	nein, Diskurs	

Tabelle 12.4. Offen ablesbare Sprechakte.

In anderen Fällen muß über die Eigenschaften der Instantiierungen bestimmt werden, welcher Sprechakt vorliegt. Die Konsistenzeigenschaft (siehe Eigenschaft (3) in Definition 4) der Instantiierungen ist hierbei hilfreich. Ist eine Instantiierung eines Interaktionsmusters nicht beendet, und die semantische Repräsentation des aktuellen Turns nicht kompatibel mit der bisher etablierten Information, wird angenommen, daß ein Unterdialog eingerichtet werden muß.

12.3.2 Auswahl von Interaktionsmustern

Die Auswahl der Interaktionsmuster erfolgt nach den Angaben in Tabelle 12.1.

12.3.3 Zusammenhang von Diskursstruktur und Dialogzielbeschreibungen

Aus dem Instantiierungsalgorithmus folgt, daß die den Dialogzielbeschreibungen zugeordneten Dienste nur dann ausgeführt werden, wenn die Instantiierung eines Interaktionsmusters *direkt unter der Wurzel* des Diskursbaumes erfolgreich beendet ist.

12.4 Implementierung

Die Instantiierung von Interaktionsmustern hängt, wie oben beschrieben, vom Systemzustand ab. Der Zusammenhang zwischen Systemzustand und Instantiierung des Interaktionsmusters ist in Tabelle 12.1 angegeben. Eine naheliegende Implementierung des Dialogsystems wäre mit Hilfe der Tabelle 12.1 in Abhängigkeit des Systemzustandes das zu instantiierende Interaktionsmuster zu bestimmen.

Zusätzlich zu den Interaktionsmustern muß das Dialogsystem jedoch auch andere Aufgaben steuern, wie zum Beispiel Datenbankanfragen durchführen, oder Eingabesätze analysieren. Dazu kommt die Verwaltung der Diskursstruktur. Aus diesem Grunde wird das Verhalten des Dialogsystems durch ein Constraint-logisches Programm (CLP) beschrieben. Das CLP hat die Aufgabe, die Repräsentationen im Diskurs mit den Wissensquellen (Dialogziele, Datenbankwächter und so weiter) zu vergleichen, und, falls nötig, entsprechende Aktionen auszulösen. Anschließend werden die Zustandsvariablen angepaßt. Nun wird bestimmt, welches Interaktionsmuster instantiiert werden soll. Nach der Instantiierung des Interaktionsmusters werden die Zustandsvariablen erneut angepaßt, die Mitteilung an den Anwender wird dem Sprachsynthesesystem übermittelt und der Kreis beginnt von vorne.

12.5 Ein Beispiel

Im folgenden wird das in Kapitel 6 gegebene Beispiel unter Berücksichtigung der Interaktionssequenzen wiederholt.

Anwender: I want to have Burritos for dinner
Repräsentation: $[\bot]$
Dialogzustand: $\langle poor, intermediate_2, speechact, selected, none, selected\rangle$
System: I am sorry I did not understand everything you said. Please say "give directions", "display" or "where am I".

Der Spracherkenner hat die Eingabe nicht richtig erkennen können. Demzufolge hat der robuste semantische Parser keine Ableitung finden können und eine leere semantische Repräsentation erzeugt. Die Information im Diskurs ist nicht ausreichend, um einen Datenbankzugriff durchführen zu können. Alle Dialogzielbeschreibungen sind mit der Information im Diskurs kompatibel. Das ausgewählte Interaktionsmuster ist das Fragen-Interaktionsmuster, welches unter Berücksichtigung der ausgewählten Dialogzielbeschreibungen dem Benutzer die Funktion des Systems auswählen läßt.

Anwender: Give directions.
Repräsentation: $[act\_givedirections]$
Dialogzustand: $\langle good, intermediate_2, sa\_answer, selected, none, determined\rangle$
System: Would you like to go to a bar, a restaurant or a museum?

Die Erkennung ist gut, ebenso konnte der Sprechakt gut erkannt werden. Es ist nach wie vor nicht genügend Information vorhanden, um einen Datenbankzugriff durchführen zu können, deswegen bleiben die Werte der Variablen Reference und ReferringExpressions unverändert. Die semantische Repräsentation schließt zwei der drei Dialogzielbeschreibungen als mögliche Repräsentation der Intention des Anwenders aus; allerdings ist noch nicht alle notwendige Information für die Dienstanforderung vorhanden, da der Zielort fehlt. Deswegen ist der Wert der Variablen Intention *determined*. Das ausgewählte Interaktionsmuster ist

wieder das Fragen-Interaktionsmuster. Die Datenbankwächter verhindern den Datenbankzugriff, da die vorhandenen Repräsentationen nicht spezifisch genug sind. Deswegen wird eine Ersetzungsfrage ausgewählt, die nach der Kategorie des Zielortes fragt. Die Ersetzungsfrage wird unter Berücksichtigung der bekannten Kategorien in eine Disjunktivfrage umgewandelt.

Anwender: A restaurant.

Repräsentation:

$$\begin{bmatrix} act\_givedirections \\ \text{PATH} \begin{bmatrix} obj\_path \\ \text{DST}\ obj\_restaurant \end{bmatrix} \end{bmatrix}$$

Dialogzustand: $\langle good, intermediate_2, sa\_answer, determined, none, determined \rangle$
System: What kind of food would you like? Please tell me the nationality.

Die Information im Diskurs ist ausreichend genug, um die Datenbanken für Bars und Museen auszuschließen, aber nicht spezifisch genug, um die Datenbankwächter für die Restaurant-Datenbank zu erfüllen. Deshalb nimmt die Variable REFERENCE den Wert *Determined* an. Alle anderen Werte bleiben unverändert. Die Situation hier ist analog zu der obigen; lediglich wird die Ersetzungsfrage nicht umgewandelt. Dies ist damit begründet, daß die hohe Anzahl der Nationalitäten die Umwandlung nicht ermöglicht.

Anwender: Mexican.

Repräsentation:

$$\begin{bmatrix} act\_givedirections \\ \text{PATH} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant \\ \text{NATIONALITY}\ prp\_mexican \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Dialogzustand: $\langle good, intermediate_2, sa\_request, finalized, none, determined \rangle$
System: *System führt Datenbankanfrage aus.*
Dialogzustand: $\langle good, intermediate_2, sa\_answer, finalized, ambiguous, determined \rangle$
System: Would you like to go to a Mexican restaurant in Oakland, Shadyside or North Hills?

Die Information ist ausreichend genug, um den Datenbankzugriff durchzuführen, folglich erhält die Variable REFERENCE den Wert *finalized* nach dem Datenbankzugriff. Der Datenbankzugriff erzeugt eine Ergebnismenge mit fünf Objekten, deswegen nimmt die Variable REFERRINGEXPRESSIONS den Wert *ambiguous* an. Alle anderen Werte bleiben unverändert. Um den Zielort zu bestimmen, wird dem Benutzer die Möglichkeit gegeben, aus Restaurants in verschiedenen Stadtteilen auszuwählen. Dies ist durch die Information aus der Datenbank gesteuert.

Anwender: Oakland.

Repräsentation:

$$\begin{bmatrix} act\_givedirections \\ \text{PATH} \begin{bmatrix} obj\_path \\ \text{DST} \begin{bmatrix} obj\_restaurant \\ \text{NATIONALITY}\ prp\_mexican \\ \text{ADDRESS} \begin{bmatrix} address \\ \text{NEIGHBORHOOD}\ \text{“}Oakland\text{”} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Dialogzustand: $\langle good, intermediate_2, sa\_answer, finalized, unique, finalized \rangle$
System: I am giving you directions to “Mad Mex“ in Oakland.

Nach Disambiguierung der Objektbeschreibungen der Restaurants verbleibt nur ein passendes Objekt, deswegen erhält die Variable REFERRINGEXPRESSIONS den Wert *unique*. Außerdem sind die Constraints der Dialogzielbeschreibung nun erfüllt, folglich nimmt die Variable INTENTION den Wert *finalized* an.

12.6 Vergleich zwischen Interaktionsmuster und Endlichen Automaten

In den vorigen Kapitel wurde wiederholt der hier beschriebene Ansatz mit auf endlichen Automaten basierenden Dialogsystemen verglichen. Ein Zustandsübergang in einem endlichen Automaten definiert eindeutig den neuen Zustand.[1]

Die Sachlage in der vorliegenden Arbeit ist jedoch eine andere. Es ist zwar richtig, daß, wie in Abschnitt 7.5 beschrieben, die Instantiierung eines Interaktionsmusters den Zielzustand teilweise bestimmt. Allerdings können nicht alle Zustandsvariablen einzig und allein aufgrund des Wissens, daß Interaktionsmuster X instantiiert ist, angepaßt werden. Aus diesem Grunde werden die Werte der Zustandsvariablen nach jedem Turn durch Inspektion der Repräsentation neu bestimmt. Damit stellt sich die Verarbeitung des Dialogs dar wie in Abbildung 12.1 gezeigt.

while(state $\notin$ accepting states)

1. play prompt;
2. accept input;
3. determine matching state transition;
4. move to new state;

end

while(state $\notin$ accepting states)

1. play prompt;
2. accept input;
3. incorporate representation in discourse;
4. request appropriate services;
5. update state variables;
6. continue current or select new interaction pattern;

end

Abbildung 12.1. (a) Verarbeitungsschritte eines Dialogsystems, daß auf endlichen Automaten beruht. (b) Verarbeitung von Instantiierungen von Interaktionsmustern

Hier ist ersichtlich, daß bei auf endlichen Automaten basierenden Dialogsystemen der gegenwärtige Zustand zusammen mit der Semantik der letzten Äußerung der Folgezustand nicht nur eindeutig bestimmt ist, sondern auch für den gegebenen Zustand und die gegebene Eingabe konstant ist. Bei der Instantiierung von Interaktionsmustern hingegen wird die Eingabe zunächst interpretiert und in den Diskurs eingefügt. Anschließend werden alle nun anforderbaren Dienste angefordert, wie zum Beispiel Datenbankzugriff. Daraufhin erst wird der abstrakte Dialogzustand *als Funktion der im Diskurs vorhandenen Information* angepaßt. Somit wird der Folgezustand beim Verarbeiten von

[1] Im Falle nicht-deterministischer Automaten, eine Menge von Zielzuständen; allerdings werden nicht-deterministische Automaten aus diesem Grunde nicht gern für Dialogsysteme verwandt.

Instantiierungen von Interaktionsmustern indirekt berechnet. Das Dialogmanagement hängt damit von der im Diskurs vorhandenen Information ab und nicht von einem beim Systementwurf festgelegten statischen Zustandsübergang.

12.7 Diskussion

Verschiedene Instantiierungen von Interaktionsmustern erlauben, Informationen im Dialog zurückzunehmen oder Antworten zu verweigern. Diese Instantiierungen werden durch die Eingaben *"Start over", "Undo", "I don't know it", "I am feeling lucky"* ausgelöst. Hierbei wird der Dialogzustand an den Anfang des Dialogs, um eine Interaktion zurück, um eine Interaktion voraus oder an das Ende des Dialoges navigiert. Bei den letzten beiden Instantiierungen wird fehlende Information zufällig ausgewählt.

Von einem anderen Standpunkt betrachtet, kann man damit im Diskurs navigieren, ganz ähnlich wie man mit den *Vorwärts* und *Zurück* Knöpfen eines Internet Browsers die *Browser History* navigieren kann. Abbildung 12.2 stellt den Zusammenhang der Navigationskommandos und dem Informationsgehalt im Diskurs dar.

Eine Präsentation der Fähigkeiten des Systems bezüglich der Navigation, ähnlich wie in Abbildung 12.2, mag hilfreich für Benutzer des Systems sein. Wie jedoch die Ausführungen zu den Interaktionsmustern schon gezeigt haben, ist die Organisation der Systemarchitektur bezüglich der Navigation nicht erwünschenswert, da die Navigationskommandos nicht bestimmte Eigenschaften des Systems enkapsulieren. Die Interaktionsmuster, wie die Ausführungen zu deren Implementierung belegt, tun dies hingegen.

Das hier vorgeschlagene Zusammenspiel von Interaktionsmustern und Dialogzuständen stellt sprach- und domänenunabhängige Prinzipien zur Dialogverarbeitung bereit. Dies gleicht insofern den Arbeiten am Dialogsystem ARTIMIS, auf die schon mehrfach verwiesen wurde [Bretier and Sadek, 1996,Sadek *et al.*, 1997]. Auch hier wurden sprach- und domänenunabhängige Prinzipien, formuliert in Modallogik, verwendet, um den Dialog zu steuern.

12.8 Zusammenfassung

In diesem Kapitel wurden die vorgestellten Interaktionsmuster nach verschiedenen Kriterien klassifiziert. Die Klassifikation in Abhängigkeit des abstrakten Dialogzustandes schränkt die Anwendbarkeit der Interaktionsmuster ein und ist somit eine Entscheidungshilfe für das Dialogsystem, welche Aktion es auszuführen hat.

Die fortlaufende Instantiierung der Interaktionsmuster induziert die Struktur des Diskurses. Hierzu – und zur Bestimmung des abstrakten Dialogzustandes – ist es notwendig, daß der Sprechakt des aktuellen Turns bstimmt wird. Ist der Sprechakt nicht offen erwähnt, werden die Constraints über die Instantiierungen der Interaktionsmuster ausgenutzt.

Das Dialogsystem benutzt die hier gezeigten Klassifikationen, um neue Interaktionsmuster zu instantiieren oder partielle Instantiierungen fortzuführen oder

zu beenden. Damit bestimmt natürlich der Umfang des Katalogs von Interaktionsmustern die Fähigkeiten des Dialogsystems. Mit dem vorliegenden Katalog von Interaktionsmustern ist es beispielsweise nicht möglich, *Counterfactuals* oder hypothetische Anfragen zu verarbeiten. Eine Erweiterung der bekannten Sprechakte und der Menge der Interaktionsmuster wäre hierzu notwendig.

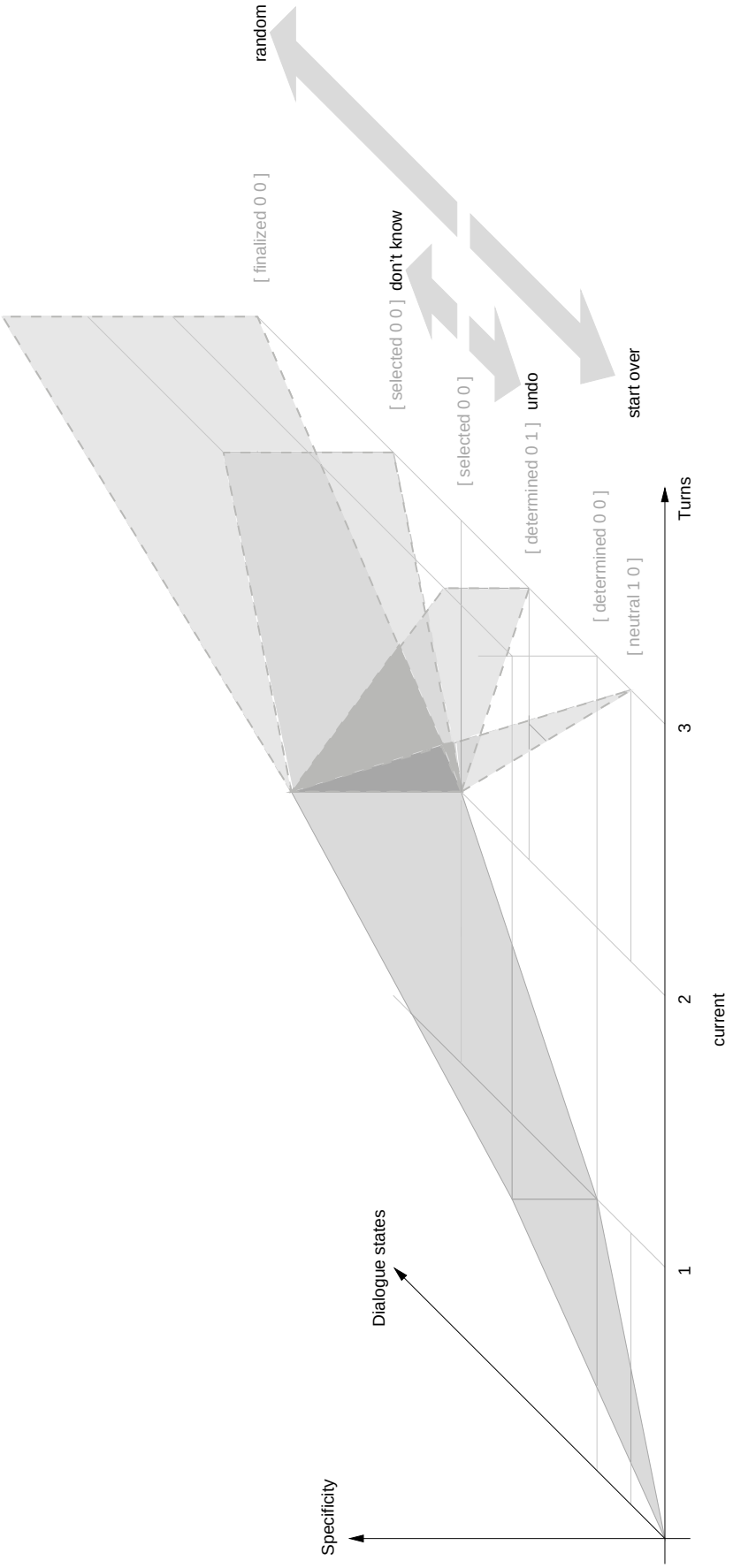


Abbildung 12.2. Navigation durch den Dialog mit den Neustart, Widerrufen, Weiss Nicht und Egal Instantiierungen von Interaktionsmustern.

Teil III

Experimente, Auswertung und Schlußwort

Kapitel 13

Evaluation

13.1 Einleitung

In diesem Kapitel werden die Experimente zur Portierbarkeit des Dialogsystems dargestellt. Ziel der Untersuchung ist zu zeigen, daß die Implementierung eines prototypischen Dialogsystems unter Benutzung der in Teil I und II beschriebenen Architektur effizient und unter Wiederverwendung bereits existierender Wissensquellen geschehen kann. Zu diesem Zwecke wurden sechs Probanden gebeten, eine Anwendung zu entwerfen, zu implementieren und mit einer Sprachsteuerung zu versehen. Der Versuchsablauf ist in 8 aufeinanderfolgende Schritte gegliedert. Die ersten beiden Schritte dienen zur Implementierung der Anwendung und der nötigen Datenbanken. In den folgenden fünf Schritten werden die für die Dialogverarbeitung nötigen Wissensquellen erstellt; dies sind die Spezifikationen von (in zeitlicher Reihenfolge) Ontologie, Dialogzielen, Datenbankkonvertierungsregeln, Parsing- und Generierungsgrammatiken.

13.2 Experiment zur Untersuchung der Portierbarkeit

13.2.1 Ziel des Experiments

Es ist Ziel dieses Experimentes zu zeigen, daß die in den Teilen I und II beschriebenen Techniken zur Modularisierung die folgenden Eigenschaften haben:

1. **Schneller Entwurf**
   Ein prototypisches System kann schnell und effizient entwickelt werden
2. **Wiederverwendbarkeit**
   Existierende Spezifikationen von Wissensquellen können wiederverwendet werden

13.2.2 Aufbau des Experiments

Der Entwurf eines Dialogsystems ist untergliedert in acht Schritte. In jedem Schritt wird ein Aspekt des zu entwerfenden Dialogsystems spezifiziert oder implementiert. Im folgenden werden die acht Schritte genauer beschrieben.

Schritt 1: Entwurf und Implementierung der Dienste der Anwendung. In diesem Schritt werden die Dienste der Anwendung entworfen und implementiert. Ein Anwendungsdienst kann im Prinzip jeder über TCP/IP angebotene Dienst sein. In diesem Versuch wurde ein Dienst jedoch beschränkt auf eine JAVA-Methode, die über einen Mechanismus, der *Remote Method Invocation* ähnelt, aufgerufen werden kann. Der Ziel dieses Schrittes ist es, die Spezifikationen der benötigten Methoden anzugeben. In einem Dialogsystem, das Telephon-Anrufe an Angestellte vermittelt, wäre das Ergebnis dieses Schrittes etwa folgende Spezifikation:

```
  /**
   * transfers the call to the given phone number
   */
public void forwardCall(String phoneNumber);

  /**
   * gives information on the person such as office location
   */
public void giveInformation(String firstName,String lastName,
                            String phoneNumber,String officeNumber);
```

Schritt 2: Entwurf und Implementierung der Datenbank. In diesem Schritt wird die Datenbank des Dialogsystems entworfen. Die Datenbank speichert Eigenschaften der Objekte in der Domäne. In der Anruf-Vermittlungsanwendung beispielsweise enthält die Datenbank Namen, Büronummern und Telephonnummern der Angestellten. Die Datenbank kann auf zwei unterschiedliche Weisen implementiert werden. Die einfachste Möglichkeit besteht darin, eine MS ACCESS Datenbank zu entwerfen, auf die das Dialogsystem dann zugreifen kann. Alternativ dazu besteht die Möglichkeit, Objekte in der Domäne durch Instanzen von JAVA-Klassen zu repräsentieren. In dieser Variante werden zum Systemstart alle referenzierbaren Objekte erzeugt und in der Anwendung registriert. Referenzierende Ausdrücke werden dann durch Referenzen auf Objekte aufgelöst.

Schritt 3: Entwurf der Ontologie. Wie in Abschnitt 2.2 beschrieben, muß das Dialogsystem über eine Ontologie verfügen, die die "verstandenen" Konzepte und die Beziehungen untereinander deklariert. Im dritten Schritt wird ein Objektmodell der vom System zu verstehenden Domäne deklariert.

Schritt 4: Entwurf der Dialogzielbeschreibungen. Aufbauend auf der im dritten Schritt entworfenen Ontologie werden im vierten Schritt die Dienste der Anwendungen mit den Repräsentationen in Beziehung gesetzt. Das Ziel des vierten Schrittes ist es, für die in Schritt 1 entworfenen Dienste die Dialogzielbeschreibungen anzugeben.

Schritt 5: Entwurf der Datenbank-Konvertierungsregeln. Semantische Repräsentationen von definition Beschreibungen müssen durch Informationen

aus den Datenbanken aufgelöst werden. Zu diesem Zweck müssen Merkmalspfade der semantischen Repräsentationen mit Datenbankfeldern in Beziehung gesetzt werden. Dies geschieht durch Angabe von Konvertierungsregeln und Datenbankwächtern, wie in Abschnitt 5.3 beschrieben. Das Ziel diesen Schrittes ist es, diese Konvertierungsregeln und Wächter anzugeben.

Schritt 6: Entwurf der Grammatiken. Grammatiken stellen die Beziehung her zwischen Ausdrücken in natürlicher Sprache und semantischen Repräsentationen. Im sechsten Schritt werden die Grammatikregeln und Konvertierungsregeln wie in Abschnitt 4.3 beschrieben spezifiziert. Zu diesem Zeitpunkt wird von den Möglichkeiten der Vererbung von Grammatikregeln noch kein Gebrauch gemacht.

Schritt 7: Entwurf der Generierungs-Schablonen. In Schritt 7 werden die Generierungs-Schablonen erstellt, mit deren Hilfe Klärungsfragen und Bestätigungen erreichter Ziele für den Anwender generiert werden.

Verwendete Hardware. Das Dialogsystem läuft auf mit PENTIUM III Prozessoren ausgestatteten Computern unter dem WINDOWS NT Betriebssystem. Als Eingabemikrofon dient ein Standard *CloseTalk* Mikrophon, wie es mit handelsüblichen Spracherkennersoftware mitgeliefert wird.

Verwendete Programme. Spracheingabe in das Dialogsystem kann durch den JANUS 4 Spracherkenner oder alternativ durch jeden Erkenner erfolgen, der MS SAPI 4 unterstützt. Das verwendete Sprachsynthesesystem ist das an der University of Edinburgh entwickelte FESTIVAL [Taylor *et al.*, 1998]. Alternativ können MS SAPI 4 unterstützende Sprachsynthesesysteme zum Einsatz kommen.

13.2.3 Ablauf des Experiments

Die Probanden erhalten zu Beginn jeden Schrittes eine Anleitung, die das Ziel des gegenwärtigen Schrittes und die zu spezifizierenden Wissensquellen beschreibt. Die Probanden haben die Möglichkeit, zu jeder Zeit Fragen zu stellen. Am Ende jeden Schrittes werden die Spezifikationen in interne Repräsentationen übersetzt, um sicherzustellen, daß die vorgeschriebene Syntax für die Spezifikationen eingehalten wird. Zusätzlich zu der Dokumentation hatten die Probanden Gelegenheit, zu jeder Zeit beliebige Fragen zu stellen und Erklärungen zu verlangen. Davon wurde insbesondere nach Lektüre der Anleitung Gebrauch gemacht.

13.2.4 Die Fallstudien

Im folgenden werden die entworfenen Dialogsysteme kurz beschrieben.

Roulette Das Roulette-Dialogsystem verfügt über eine Datenbank von möglichen Roulette-Spielern. Die Dialogziele ermöglichen es, für jeden Spieler auf eine Zahl, eine Farbe oder gerade oder ungerade zu setzen, oder das Spiel zu starten. Die verwendete Sprache des Systems ist deutsch.

Video-Verleihsystem Das Video-Verleihsystem ermöglicht es, sich zu erkundigen, ob ein bestimmter Videofilm erhältlich ist, und gegebenfalls den Film auszuleihen. Die Datenbank enthält Informationen über Schauspieler, Sprache der Filme, und Speichermedium (Video oder DVD). Die verwendete Sprache des Systems ist deutsch.

Anrufvermittler-System Das Anrufvermittlersystem ermöglicht es, Büronummern und Telephonnummern der Mitarbeiter des ISL labs an der Carnegie Mellon University abzurufen oder einen Telephonanruf weiterzuvermitteln.[1] Die Datenbank enthält Informationen über die Mitarbeiter. Die verwendete Sprache des Systems ist englisch.

Eintrittskartenverkauf Dieses System ermöglicht es, Eintrittskarten für den $KennywoodAmusementPark$ in Pennsylvania zu erstehen (simuliert). Die Datenbank besteht aus Informationen über die Eintrittskarten. Die verwendete Sprache ist englisch.

Bierinformationssystem Dieses System ermöglicht es, Informationen zu verschiedenen Bieren zu erfragen. Die Informationen über die Biere sind in der Datenbank gespeichert. Die verwendete Sprache ist englisch.

Steuerung eines CD-Spielers

Mit diesem System ist es möglich, einen CD-Spieler per Spracheingabe zu bedienen. Die verwendete Sprache dieses Systems ist englisch.

13.2.5 Auswertung der Fallstudien

Es ist Ziel dieses Experimentes zu zeigen, daß in kurzer Zeit prototypische Dialogsysteme nach den Spezifikationen der Probanden erstellt werden können. Von daher ist die Zeitdauer der Implementierung ein wichtiger Aspekt. Die erstellten Dialogsysteme variieren jedoch stark in der Komplexität der Grammatiken, Aufgaben- und Domänenmodelle, so daß diese Aspekte ebenfalls mit in Betracht gezogen werden müssen. Aus diesem Grunde wird im folgenden ein Komplexitätsmodell entwickelt, mit dem der Grad der Komplexität der Spezifikationen gemessen werden kann. Desweiteren ist ein wesentlicher Punkt, inwieweit die Probanden in der Lage waren, existierende Spezifikationen wiederzuverwenden.

[1] Dieses Verhalten ist simuliert, kann aber durch entsprechende Programmierung einer entsprechenden PC-Erweiterungskarte, wie beispielsweise von DIALOGIC erhältlich, ohne weiteres implementiert werden.

Auswertung der erstellten Spezifikationen Die Tabelle 13.1 stellt die Größe der spezifizierten Wissensquellen dar. Die hier angegebenen Zahlen beziehen sich lediglich auf die von den Probanden erstellten Wissensquellen und schließen nicht die wiederverwendeten Wissensquellen mit ein.

Diese Tabelle spiegelt die unterschiedliche Komplexität der Anwendungen wider. Das Bier-Informationssystem hat ein komplexes Domänenmodell, mit verschiedenen Typen für die entsprechenden Biere. Merkmale beschreiben Geschmacksrichtungen und Herkunftsland. Auf der anderen Seite hat das Anrufvermittler-System ein kleines Domänenmodell, das lediglich die beiden Aktionen (Anruf weiterleiten und Telephonnummer bekanntgeben) sowie den Arbeitnehmer modellieren muß.

Auswertung der wiederverwendeten Spezifikationen Die Tabelle 13.2 stellt den Prozentanteil des wiederverwendeten Domänenmodells dar.

Der hohe Grad der Wiederverwendbarkeit von Typen liegt daran, daß alle Objekte, Aktionen und Eigenschaften in den Domänen von bereits existierenden Typen abgeleitet werden konnte.

Anteil der automatisch generierten Grammatikregeln Die Tabelle 13.3 stellt den Anteil der automatisch generierten strukturellen Grammatikregeln dar.

Die erzeugten strukturellen Regeln haben im wesentlichen zwei Ursprünge. Zum einen stammen sie aus der Vererbungshierarchie, um partielle Information zu simulieren. Das heißt, das die Anzahl der strukturellen Regeln mit der Größe des Domänenmodells wächst. Dies ist insbesondere an den beiden Extremfällen, dem Bier-Informationssystem und dem Anrufvermittler-System zu sehen. Auf der anderen Seite kann durch die Verwendung von abstrakten Basisregeln der Aufwand des Grammatikschreibens eingespart werden. Hier haben beispielsweise das Roulette-System und das CD-Spieler System Vorteile gegenüber dem Video-Verleihsystem, dem Eintrittskartenverkaufssystem und Anrufvermittler-System, bei etwa gleicher Größe des Domänenmodells.

In Tabelle 13.4 ist die verwendete Zeit für die Erstellung der Spezifikationen aufgeführt. Die angegebenen Zeiten beinhalten lediglich die Zeit, die am Rechner verbracht worden ist, um die Spezifikationen zu erstellen, nicht jedoch die Zeiten für Erklärungen, Fragen und Lesen der Dokumentation. Die Zeiten für Schritte 1 und 2 werden nicht angegeben, da in diesen Schritte die Anwendung selbst spezifiziert wird. Es soll hier nur über den für das Dialogsystem verwendeten Zeitaufwand berichtet werden.

13.2.6 Auswertung der Grammatik-Hilfsmechanismen

Die Effektivität der Grammatikhilfsmittel wurde ebenfalls in einem mittelgroßen System verglichen. Hierzu wurden semantische Standard-Grammatiken, die für den PHOENIX-Parser entwickelt worden sind, mit den neuen vektorisierten Gramamtiken verglichen. Die für den PHOENIX-Parser entwickelten Grammatiken sidn im Rahmen des VODIS-Projektes entstanden und decken Autonavigationsanfragen ab. Die vektorisierten Grammatiken sind im Rahmen eines

Anwendung	**Initialen**	**Domänenmodell**	**Aufgabenmodell**	**Grammatikregeln**	**Datenbankregeln**	**Schablonen**	**Dateilänge**
	(der Versuchsperson)	Typen/Merkmale	Dialogzielbeschreibungen	lexikalisch/strukturell	Tabellen / Felder	Fragen / Aussagen	Zeilen
Bier-Informationssystem	rgm	193/24	2	256/200	1/5	5/2	456
Anrufvermittler-System	msb	88/26	2	77/39	1/8	4/2	163
Eintrittskartenverkauf	dw	89/22	3	11/50	1/ 3	4/4	236
Roulette	ah	96/29	4	129/65	1 /3	9/4	349
CD-Spieler	akj	108/29	6	106/66	1/4	7/0	308
Video-Verleihsystem	kp	95/22	1	236/72	2/5	4/1	330

Tabelle 13.1. Größe der erstellten Wissensquellen

Anwendung	**Domänenmodell** Typen	**Domänenmodell** Merkmale
Bier-Informationssystem	70 %	70 %
Anrufvermittler-System	95 %	57 %
Eintrittskartenverkauf	94 %	77 %
Roulette	87 %	59 %
CD-Spieler	79 %	59 %
Video-Verleihsystem	90 %	77 %

Tabelle 13.2. Größe der wiederverwendeten Wissensquellen

Anwendung	**Absolut**	**Relativ**
Bier-Informationssystem	169	84 %
Roulette	26	40 %
CD-Spieler	25	38 %
Anrufvermittler-System	9	23 %
Video-Verleihsystem	17	23 %
Eintrittskartenverkauf	9	18 %

Tabelle 13.3. Größe der automatisch generierten Grammatikregeln

von General Motors geförderten Projektes entstanden und decken dieselben Anfragen wie die Phoenix-Gramamtiken ab.

Wie aus der Tabelle 13.5 ersichtlich ist, können im Falle der vektoriellen Grammatiken etwa ein Drittel der benötigten Regeln automatisch erzeugt werden und entlasten somit den Grammatikschreiber. Insbesondere kann die Ververbungsbeziehung zwischen den Typen in der Typenhierarchie ausgenutzt werden, um automatisch Regeln zu erzeugen. Es soll bemerkt werden, daß – obwohl beim Erstellen der Grammatik ohne Hilfsmittel das Konzept der Vererbungsregel unbekannt war – etwa ein Fünftel aller Regeln Vererbungsregeln waren. Damit stellen sie einen bedeutenden Teil der zu erstellenden Grammatiken dar. Im Falle der vektorisierten Grammatiken können diese Regeln automatisch erzeugt werden. Darüber hinaus konnten auch strukturelle Regeln durch die Vererbungstechniken automatisch erstellt werden.

Die Fallstudie Das wichtigste Ergebnis dieser Fallstudie ist, daß innerhalb kurzer Zeit ein funktionierender Prototyp von den Probanden erstellt werden konnte. Die längste Zeitdauer liegt mit etwas über acht Stunden leicht über der Dauer eines Arbeitstages.

Als weiterer Punkt ist zu vermerken, daß der Hintergrund der Probanden sehr unterschiedlich war. Teilweise waren objekt-orientierte Konzepte nicht bekannt, oder Programmierkenntnisse nur rudimentär vorhanden. Auf der anderen Seite gab es Probanden, die kaum Kenntnisse von Grammatiken hatten.

Teilnehmer mit wesentlicher Erfahrung im Grammatikschreiben hatten Schwierigkeiten, sich an das Konzept der Vererbung von Grammatikregeln zu gewöhnen. Mit der Wiederverwendung von existierenden Grammatikregeln wurde ein Verlust der Kontrolle über die Grammatik verbunden. Auf der an-

Anwendung	**Schritt 3**	**Schritt 4**	**Schritt 5**	**Schritt 6**	**Schritt 7**	**Summe**
Bier-Informationssystem	30 min	15 min	40 min	10 min	70 min	165
Anrufvermittler-System	60 min	20 min	60 min	20 min	60 min	220
Eintrittskartenverkauf	15 min	5 min	10 min	60 min	60 min	150
Roulette	30 min	30 min	30 min	30 min	45 min	165
CD-Spieler	190 min	130 min	130 min	10 min	60 min	520
Video-Verleihsystem	75 min	30 min	120 min	40 min	120 min	385

Tabelle 13.4. Zeitlicher Aufwand für die einzelnen Schritte

Anwendung	Zielsprache	Regeln			Summe
		lexikalisch	strukturell	Vererbungs-	
ohne Hilfsmittel	englisch	803	293	202	1.298
davon automatisch		0	0	0	0
mit Hilfsmitteln	englisch	570	124	281	975
davon automatisch		0	24	281	305

Tabelle 13.5. Vergleich zwischen semantischen Standard-Grammatiken und erweiterten Gramamtiken. Lexikalische Regeln, die zu Eigennamen expandieren (wie beispielsweise Straßennamen oder Restaurantnamen), sind in dieser Statistik nicht aufgeführt.

deren Seite wurde von zwei Teilnehmern der Studie, die beide über keine Erfahrungen mit Grammatiken, aber über Programmiererfahrung verfügen, das Prinzip der Vererbung von Grammatikregeln richtig angewandt; und zwar zu einem Zeitpunkt, als es noch nicht in der Dokumentation erwähnt worden ist. Diese beiden Teilnehmer waren in der Lage, die korrekte Anwendung aus dem Beispiel abzuleiten.

Weiterhin ist zu vermerken, daß die implizite Kontrolle über den Dialogverlauf durch die Spezifikationen der Wissensquellen einiger Erklärung bedurfte. Es war nicht immer sofort verständlich, daß beispielsweise Datenbankzugriffe vom Dialogsystem selbständig vorgenommen werden und sich der Systemdesigner darum nicht zu kümmern braucht.

Die Interaktionsmuster In allen beschriebenen Fällen konnten die Interaktionsmuster wiederverwendet werden. Das heißt, daß in allen Dialogsystemen Klärungsfragen, Korrekturen und in einigen Fällen auch das Anpassen von Beschreibungen korrekt implementiert worden ist. Dies zeigt die Allgemeingültigkeit der Interaktionsmuster.

13.3 Vollständige Evaluierung eines Gesamtsystems

13.3.1 Ziel des Experiments

Ziel des Experiments ist zu zeigen, daß nicht nur einfache Dialoguesysteme schnell entworfen werden können, sondern daß die im Rahmen dieser Arbeit entwickelten Techniken auch komplexere Dialogverarbeitung unterstützen.

13.3.2 Aufbau des Experiments

Das Stadtinformationssystem wurde verwendet, um die Gesamtperformanz des Systems zu evaluieren. Das System wurde auf einem Rechner mit zwei Pentium III Prozessoren (500 MHz) unter dem Windows 2000 Betriebssystem installiert. Der verwendete Spracherkenner ist Janus 4.0. Als Eingabemikrofon diente ein handelsübliches Mikrofon, das mit gängiger Spracherkennersoftware mitgeliefert wird.

13.3.3 Durchführung des Experiments

Die Versuchspersonen wurden in das System eingewiesen. Zunächst wurde eine Beispielinteraktion vorgeführt, anschließend hatten die Versuchspersonen Gelegenheit, die Beispielinteraktion zu wiederholen. Im folgenden wurde dann der vom System anzufordernde Dienst auf einer Karteikarte stichwortartig beschrieben. Ein Beispiel einer solchen Beschreibung ist in Abbildung 13.1 gezeigt.

Die Anzahl der auf der Karte angegebenen Konzepte reicht von eins bis fünf.

Action	directions
Type	Restaurant
Property	mexican
Location	Oakland

Abbildung 13.1. Beispiel einer Aufgabenbeschreibung. In diesem Falle sind vier anzufordernde Konzepte gegeben.

13.3.4 Evaluierung

Es wurden insgesamt 75 Dialoge durchgeführt. Die Verteilung der Anzahl der auf den Karteikarten angegebenen Konzepte ist in Tabelle 13.6 angegeben.

Anzahl gegebene Konzepte	1	2	3	4	5	Summe
Anzahl Dialoge	13	16	30	12	4	75

Tabelle 13.6. Verteilung der Anzahl der angeforderten Konzepte.

Konzeptgenauigkeit Im folgenden wird die Konzeptgenauigkeit angegeben. Hierbei bedeutet *erkanntes Konzept*, daß das infragestehende Konzept vom Spracherkenner erkannt, und vom Parser korrekt in die semantische Repräsentation eingefügt worden ist.

Anzahl gegebene Konzepte	1	2	3	4	5	Summe
Anzahl Dialoge	13	16	30	12	4	75
Anzahl vermittelte Konzepte	32	85	171	81	28	397
Anzahl erkannte Konzepte	13	56	126	56	24	275
Konzeptgenauigkeit	40.1%	65.9%	73.7%	69.1%	85.7%	69.3%

Tabelle 13.7. Verteilung der Anzahl der angeforderten Konzepte.

Anzahl gegebene Konzepte	1	2	3	4	5	Summe
Anzahl Dialoge	13	16	30	12	4	75
minimale Länge	1	1	1	1	3	1
durchschnittliche Länge	2,5	3,6	3,4	3,8	4	3.37
maximale Länge	5	8	11	8	6	11

Tabelle 13.8. Verteilung der Länge der Dialoge.

Länge des Dialogs Im folgenden wird die Länge der Dialoge in Anzahl Äußerungen des Benutzers angegeben.

Bestätigung der erfolgten Eingabe Um den Benutzer den Erfolg oder Mißerfolg des fortschreitenden Dialogs zu vermitteln, ist es wichtig zu bestätigen, welcher Teil der dem System übermittelten Information verstanden worden ist. Der folgende Dialog soll dies erläutern.

Benutzer I would like to go to a French restaurant in Shadyside.
System I found 4 restaurants in Shadyside, Millvale, South Side and East Liberty. Which neighborhood would you like to go to?

Der Benutzer versucht, dem System vier Konzepte zu übermitteln (nämlich *Fahrtanweisung, Restaurant, französisch* und *Shadyside*). Das System versteht drei davon (*Fahrtanweisung, Restaurant, französisch*). Der Benutzer kann inferieren, daß das System den Stadtteil nicht verstanden hat, da eine Klärungsfrage nach dem Stadtteil gestellt wird. Das Konzept *Fahrtanweisung* wird implizit durch die Frage (*... would you like to go to?*) bestätigt, das Konzept *Restaurant* durch den Satz *I found 4 restaurants*. Die Tatsache, daß das System das Konzept *französisch* ebenfalls verstanden hat, wird nicht bestätigt. Zusammenfassend halten wir fest, daß von vier übermittelten Konzepten drei verstanden worden sind. Dem Benutzer wird bestätigt, daß zwei Konzpte verstanden worden sind, und daß ein weiteres Konzept nicht verstanden worden ist. Die Nachricht des Systems ist in diesem Beispiel von daher suboptimal.

Um die Effektivität der Nachrichten des Systems zu evaluieren, wurden die bestätigten Konzepte mit den übermittelten Konzepten in Beziehung gesetzt. Tabelle 13.9 zeigt das Ergebnis.

	erkannt		**nicht erkannt**		**Summe**	
bestätigt	275	69.3%	118	29.7%	393	99.0%
nicht bestätigt	3	0.7%	1	0.3%	4	1.0%
Summe	278	70.0%	119	30.0%	397	100.0%

Tabelle 13.9. Absolute und relative Anzahl der bestätigten Konzepte. Die linke Spalte bezieht sich auf die korrekt erkannten Konzepte, die rechte Spalte auf die nicht erkannten (oder fehlerkannten) Konzepte.

Hier ist ersichtlich, daß von 397 übermittelten Konzepten dem Benutzer in 393 (oder 99.0% der Fälle) die Erkennung bestätigt worden ist. Damit kann man sagen, daß das Dialogsystem den Benutzer über den internen Zustand gut informiert hält.

13.3.5 Komplexere Beispieldialoge

In diesem Abschnitt werden komplexere Beispieldialoge beschrieben, die ebenfalls durch Interaktionsmuster ausgedrückt werden können.

Gegenfragen. Im folgenden soll gezeigt werden, wie das Dialogsystem und der Benutzer durch Gegenfragen die Dialogsteuerung übernehmen können. Wir betrachten den folgenden Dialog:

1 User: Give me directions to Mad Mex
2 System: There are two places called Mad Mex. Would you like to go to the Mad Mex on Atwood Street or McKnight Road?
3 User: Where is McKnight Road?
4 System: McKnight Road goes through Ross Township and North Hills.
5 User: How about Atwood Street?
6 System: Atwood Street is in Oakland.
7 User: Let's go to Oakland then.
8 System: I am giving you directions to Mad Mex in Oakland.

Die Hauptschwierigkeit in diesem Dialog ist es für das System zu erkennen, daß die Äußerungen (3) und (5) des Benutzers keine Antworten auf die gestellten Fragen darstellen. Da die semantischen Repräsentationen von (3) und (5) mit denen zu den gegebenen Zeitpunkten aktiven Dialogzielbeschreibungen inkompatibel, aber mit bereits ausgeschlossenen Dialogzielbeschreibungen kompatibel sind, wird von einem Unterdialog ausgegangen. Hier wird der gesamte Dialogzustand auf einen Stack gebracht, der Unterdialog verarbeitet, und bei Abschluß wieder vom Stack geholt.

Eine weitere Schwierigkeit besteht darin, daß zur Bestimmung des Unterdialogs in (5) zunächst die durch *"How about..."* eingeführte Ellipse aufgelöst werden muß. Schließlich sei bemerkt, daß die Antwort (7) strenggenommen nicht die Frage (2) beantwortet. Allerdings ist die Information in der Lage, die ambige Referenz des Namens Mad Mexäufzulösen. Dies ist durch die unifikationsbasierte Dialogverarbeitung begründet.

Ein weiteres Beispiel ist das folgende.

1 User: Give me directions to an Asian restaurant in Squirrel Hill.
2 System: There are five Asian restaurants in Squirrel Hill. Would you like to go to a Chinese, Japanese or Thai place?
3 User: Is there no other possibility?
4 System: There are two Cambodian restaurants in downtown. Would you like to go to Lemongrass Cafe or Phnom Penh Restaurant?
5 User: Phnom Penh Restaurant.
6 System: I am giving you directions to Phnom Penh Restaurant in Downtown.

In diesem Beispiel wird eine erneute Datenbankanfrage getriggert, bei der die Constraints, ähnlich wie bei der ANPASSEN Instantiierung, aufgeweicht werden. Von der Ergebnismenge der zweiten Datenbankanfrage wird die erste Ergebnismenge abezogen, um der Bedeutung der Nominalphrase *"other possibilities"* gerecht zu werden.

Diese Beispiele zeigen, wie durch Gegenfragen der Benutzer den Dialog beeinflussen kann. Ein weiteres Beispiel soll zeigen, daß auch das Dialogsystem Gegenfragen initiieren kann.

1 User: Give me directions to a French restaurant in Shadyside
2 System: Would you prefer to go to a French restaurant in South Side instead? This is much closer.
3 User: Okay!
4 System: I am giving you directions to Le Pommier in South Side.

In diesem Dialog gibt es eine vom Benutzer gewünschte Lösung, nämlich das französische Restaurant in Shadyside. Das Problem in diesem Dialog ist für das System zu erkennen, daß eine ähnliche Lösung, nämlich das Restaurant in South Side eventuell besser wäre. Zu diesem Zwecke werden, ähnlich wie bei der ANPASSEN Instantiierung, die Datenbank Constraints aufgeweicht, um eine größere Ergebnismenge von Restaurants zu erhalten. Die Elemente der Ergebnismenge wird dann mit einer anwendungsspezifischen Bewertungsfunktion bewertet, und die Objektbeschreibung mit der höchsten Bewertung ausgewählt. Wird durch diese Objektbeschreibung ein anderes Objekt als das ursprünglich intendierte Objekt beschrieben, wird die entsprechende Gegenfrage gestellt. Es ist offensichtlich, daß dieses Verfahren genau dann domänenunabhängig ist, wenn die Bewertungsfunktion durch die Anwendung implementiert werden kann. Dies geschieht durch die im Kapitel 5 beschriebenen Dienstanforderungen. Auf Ebene der Dialogverarbeitung jedoch ergibt sich ein größeres Problem, nämlich zu bestimmen, wann das Dialogsystem die Gegenfrage stellen soll. Werden Gegenfragen zu häufig gestellt, wirkt es störend, werden sie zu wenig gestellt, ergeben sich eventuell für den Benutzer suboptimale Lösungen. Es steht zu vermuten,

daß eine enge Integration der Dialogstrategie mit einem Benutzermodell die Qualität des Dialogs an dieser Stelle entscheidend verbessern kann.

Partielle oder komplementäre Information In den folgenden Dialogen soll gezeigt werden, wie das Dialogsystem Information verarbeiten kann, nach der nicht explizit gefragt worden ist.

1 User: Give me directions to a restaurant on Liberty Avenue
2 System: How about an Australian restaurant?
3 User: Yes sounds good. They should serve Crocodile Jerky.
4 System: I am giving you directions to Aussie's Downunder in Bloomfield.

Die Zusatzinformation, welche Gerichte das Restaurant führen soll, dient dazu, die ambige Referenz der Nominalphrase *"restaurant on Liberty Avenue"* aufzulösen. Auf Seiten der Datenbank ist diese Information in einer Zusatztabelle gespeichert, die mit der Restauranttabelle verknüpft wird. Ein weiteres Beispiel soll dies verdeutlichen.

1 User: Give me directions to a restaurant.
2 System: What kind of food should the restaurant offer?
3 User: Modern French cuisine.
4 System: There are two French restaurants that offer modern dishes. Would you like to go to a moderately expensive or an expensive place?
5 User: Moderately expensive.
6 System: I am giving you directions to Le Pommier in South Side.

Hier wird in der Antwort (3) Information für zwei Attribute vermittelt, nämlich Nationalität und Art der servierten Gerichte.

13.4 Zusammenfassung

In diesem Kapitel wurde ein Experiment zum schnellen Erstellen von Dialogsystemen beschrieben. Die Teilnehmer der Studie waren in der Lage, anhand kurzer Dokumentation und eines Beispiels ein einfaches Dialogsystem zu erstellen und zu testen. Alle Dialogsysteme konnten (zusammengenommen) innerhalb von etwa einem Arbeitstag fertiggestellt werden. Die Interaktionsmuster, gesteuert durch die von den Teilnehmern spezifizierten Wissensquellen, haben dabei die Kontrolle über den Dialog übernommen.

Kapitel 14

Schlußfolgerungen

In dieser Ausarbeitung wurde ein allgemeines Verfahren zum Erstellen von Dialogsystemen vorgestellt. Dieses Verfahren basiert auf einer dreischichtigen Systemarchitektur. Die erste Schicht implementiert grundlegende Dialogdienste, und nimmt dafür zielsprach- und domänenabhängige Wissensquellen in Anspruch. Darauf aufbauend steuern die Interaktionsmuster die Interaktion des Systems mit dem Benutzer und greifen hierzu auf die Dienste der unteren Schicht zu. Die Interaktionsmuster sind domänen- und zielsprachunabhängig und müssen nicht an ein System angepaßt werden.

14.1 Beiträge

Die in dieser Arbeit gemachten Beiträge können in verschiedenen Bereiche unterteilt werden.

14.1.1 Der Katalog von Interaktionsmustern

Der wesentliche Beitrag dieser Arbeit besteht aus den miteinander verzahnten Konzepten des abstrakten Dialogzustandes und der generischen Interaktionsmuster. Die Struktur eines Dialogsystems ist typischerweise durch die Systemkomponenten gegeben, wie Satzanalyse, Generierung und Diskursmodul. Diese Unterteilung ist aber funktional; sie beschreibt nicht die Fähigkeiten des Dialogsystems bezüglich der Sprachverarbeitung.

Im Falle von Dialogsystemen ist es aber so, daß jeder Eingabe auch eine entsprechende Implementierung entgegengesetzt werden muß – das System soll ja schließlich tun, was man ihm sagt. Aus diesem Grunde erschien es zweckmäßig, das System anhand dieser Dimension zu unterteilen. Aus dieser Überlegung entstand das Konzept des Interaktionsmusters.

Ein Interaktionsmuster beschreibt, wie die Information im Diskurs im fortschreitenden Dialog erweitert wird. Das Interaktionsmuster hat dabei Freiheiten, Informationen vom Anwender auf verschiedene Arten und Weisen zu erfragen. Damit trennen Interaktionsmuster das *was* das Dialogsystem kann, von dem *wie* es die Fähigkeiten implementiert.

Die Entscheidung, welches Interaktionsmuster an einem gegebenen Zeitpunkt zu instantiieren ist, wird in Abhängigkeit des abstrakten Dialogzustandes

getroffen. Der abstrakte Dialogzustand charakterisiert den Verlauf des Dialogs vom Anfang bis zum aktuellen Zeitpunkt bezüglich domänen- und zielsprachunabhängiger Attribute, aufgrund derer die Instantiierungen der Interaktionsmuster ausgewählt werden können.

Der hier angegebene Katalog von Interaktionsmustern ist sicherlich nicht vollständig und bedarf zukünftiger Forschung. Das vorliegende System ist beispielsweise nicht in der Lage, hypothetische und *counterfactual* Anfragen zu verarbeiten. Die Schwierigkeit bei *counterfactuals* ist, daß erst am Ende des Diskurses Informationen als wahr im Diskurs akzeptiert wird. Als Beispiel dient der folgende Dialog:

Anwender: Wie lange dauert es von hier bis zur Uni?
System: Die Fahrtzeit beträgt 25 Minuten.
Fahrtstrecke wird angezeigt
Anwender: Wie lange würde es dauern, wenn ich durch die Luisenstraße fahren würde?
System: Die Fahrtzeit würde dann 30 Minuten betragen
Neue Fahrtstrecke wird angezeigt

An dieser Stelle müssen zwei Alternativen offen gehalten werden. Der Anwender kann jetzt zwischen beiden Alternativen auswählen, und nur die Fahrtstrecke der gewählten Alternative darf auf der Karte angezeigt bleiben. Dies ist ein Beispiel für ein neues Interaktionsmuster, das vom vorliegenden System nicht abgedeckt wird.

14.1.2 Unterstützende Beiträge

Um die Interaktionsmuster und die Basis-Dialogdienste implementieren zu können, wurden verschiedene bekannte Techniken und Verfahren erweitert.

Erweiterungen der typisierten Merkmalsstrukturen. Die von HPSG her bekannten typisierten Merkmalsstrukturen wurden auf partiell unifizierte und unterspezifizierte Merkmalsstrukturen erweitert, um somit die Basis für sprach- und domänenunabhängige Klärungsfragen- und Korrekturalgorithmen zu legen. Meta-Information über den Dialog kann durch multidimensionale Merkmalsstrukturen, ebenfalls eine Erweiterung der typisierten Merkmalsstrukturen, erfolgen. Schließlich ist noch zu erwähnen, daß objekt-orientierte Merkmalsstrukturen eine Verallgemeinerung der Standard-Merkmalsstrukturen darstellen, mit denen es möglich ist, die Anforderung von Diensten der Anwendung von der vorhandenen Information abhängig zu machen.

Vererbung und Grammatiken. Die Konzepte zum Wiederverwenden von Grammatikfragmenten erlauben es, Grammatiken schneller zu schreiben.

Systemarchitektur. Die abstrakten Dialogzustände und die generischen Interaktionsmuster sind Teil der zweiten Schicht einer dreischichtigen Dialogsystemarchitektur. Die untere Schicht besteht aus den Basis-Dialogdiensten, die von modularisierten Wissensquellen gesteuert werden. Diese Aufteilung der Funktionalität in Schichten, die wiederum in semantisch zusammenhängende Module

unterteilt werden können, ist Grundvoraussetzung für das *Rapid Prototyping* von Dialogsystemen. Zum einen können protoypische Anwendungen entwickelt werden, ohne daß der Entwickler sich mit Dialogstrategien auseinandersetzen muß. Dies ist begründet in der Tatsache, daß nur die Wissensquellen der unteren Schicht neu spezifiziert werden müssen. Zum anderen können auch auf der unteren Schicht existierende Module wiederverwendet werden.

Implementierung. Die beschriebenenen Konzepte wurden in einem Dialogsystem implementiert und getestet. Es hat sich gezeigt, daß die beschriebenen Verfahren allgemein genug sind, daß verschiedene Systeme von mehreren Personen mit unterschiedlichem Wissenstand in der Dialogverarbeitung implementiert werden können.

14.2 Zukünftige Arbeiten

Zukünftige Arbeiten liegen in der Erweiterung des Katalogs der Interaktionsmuster, wie oben erwähnt. Es wäre sicherlich auch interessant zu untersuchen, wie Interaktionsmuster mit anderen Diskurstheorien, zum Beispiel Discourse Representation Theory oder Rhetorical Structure Theory zusammenhängen und sich gegenseitig ergänzen.

Teil IV

Anhang

Anhang A

Ausdrucke der erstellten Dialogsystem-Spezifikationen

A.1 Roulette

Der folgende Ausdruck zeigt die Spezifikation des Bier-Informationssystems.

```
package beer;

server: ExamplePackage localhost 5454;

import generic.dlm;
import nationalities.dlm;

module beer ExamplePackage {

  desc prop_beer_taste inherits property;

  desc prop_beer_sweet inherits prop_beer_taste;

  desc prop_beer_very_sweet inherits prop_beer_sweet;

  desc prop_beer_bitter inherits prop_beer_taste;

  desc prop_beer_very_bitter inherits prop_beer_bitter;

  desc prop_beer_medium_taste inherits prop_beer_taste;

  desc prop_beer_body inherits property;

  desc prop_beer_light inherits prop_beer_body;

  desc prop_beer_very_light inherits prop_beer_light;

  desc prop_beer_full inherits prop_beer_body;

  desc prop_beer_very_full inherits prop_beer_full;

  desc prop_beer_medium_body inherits prop_beer_body;

  desc obj_beer inherits object {
    string          : name;
    prp_nationality : origin;
    prop_beer_taste : taste;
    prop_beer_body  : body;
    bool            : fruit;
  };

  desc obj_ale inherits obj_beer;

  desc obj_americanAle inherits obj_ale {
    false : fruit;
  };

  desc obj_stout inherits obj_ale {
    false          : fruit;
    prop_beer_full : body;
  };

  desc obj_dryStout inherits obj_stout {
    prop_beer_bitter    : taste;
    prop_beer_very_full : body;
```

```
};

desc obj_oatmealStout inherits obj_stout {
  prop_beer_sweet     : taste;
  prop_beer_very_full : body;
};

desc obj_imperialStout inherits obj_stout {
  prop_beer_sweet : taste;
  prop_beer_very_full : body;
};

desc obj_sweetStout inherits obj_stout {
  prop_beer_sweet : taste;
};

desc obj_paleAle inherits obj_ale {
  false            : fruit;
  prop_beer_bitter : taste;
};

desc obj_brownAle inherits obj_ale {
  false                  : fruit;
  prop_beer_medium_taste : taste;
};

desc obj_oldAle inherits obj_ale {
  false               : fruit;
  prop_beer_bitter    : taste;
  prop_beer_very_full : body;
};

desc obj_barleywine inherits obj_ale {
  false               : fruit;
  prop_beer_bitter    : taste;
  prop_beer_very_full : body;
};

desc obj_scotchAle inherits obj_ale {
  false               : fruit;
  prop_beer_very_sweet : taste;
  prop_beer_full       : body;
};

desc obj_irishAle inherits obj_ale {
  false                  : fruit;
  prop_beer_medium_taste : taste;
  prop_beer_light        : body;
};

desc obj_belgianAle inherits obj_ale {
  false                : fruit;
  prop_beer_bitter     : taste;
  prop_beer_medium_body : body;
};


desc obj_goldenAle inherits obj_belgianAle;

desc obj_trappistAle inherits obj_belgianAle;

desc obj_altbier inherits obj_ale {
  false                  : fruit;
  prop_beer_medium_taste : taste;
  prop_beer_medium_body  : body;
};

desc obj_indiaPaleAle inherits obj_paleAle {
  prop_beer_very_bitter : taste;
  prop_beer_medium_body : body;
};

desc obj_porter inherits obj_stout;

desc obj_lager inherits obj_beer {
  false : fruit;
};

desc obj_marzen inherits obj_lager {
  prop_beer_medium_taste : taste;
  prop_beer_medium_body  : body;
};

desc obj_pilsner inherits obj_lager {
  prop_beer_light : body;
};

desc obj_lightAmericanPilsner inherits obj_pilsner {
  prop_beer_medium_taste : taste;
  prop_beer_very_light   : body;
```

```
};

desc obj_premium inherits obj_lightAmericanPilsner;

desc obj_subpremium inherits obj_lightAmericanPilsner;

desc obj_superpremium inherits obj_lightAmericanPilsner;

desc obj_dortmunderExport inherits obj_lager {
  prop_beer_medium_taste : taste;
  prop_beer_medium_body  : body;
};

desc obj_bock inherits obj_lager {
  prop_beer_bitter : taste;
  prop_beer_full   : body;
};

desc obj_dopplebock inherits obj_bock {
  prop_beer_very_full : body;
};

desc obj_steamBeer inherits obj_ale, obj_lager {
  prop_beer_medium_body : body;
  prop_beer_bitter      : taste;
};

desc obj_wheatAle inherits obj_ale {
  prop_beer_light : body;
};

desc obj_belgianWheat inherits obj_wheatAle;

desc obj_weisse inherits obj_wheatAle {
  prop_beer_bitter : taste;
};

desc obj_berlinerWeisse inherits obj_weisse {
  prop_beer_bitter : taste;
};

desc obj_lambic inherits obj_wheatAle;

desc obj_fruitLambic inherits obj_lambic {
  prop_beer_very_sweet : taste;
  true                 : fruit;
};

desc obj_raspberryLambic inherits obj_fruitLambic;

desc obj_peachLambic inherits obj_fruitLambic;

desc obj_cherryLambic inherits obj_fruitLambic;

desc obj_fruitWheatAle inherits obj_wheatAle {
  prop_beer_bitter : taste;
  true             : fruit;
};

desc obj_cherryWheat inherits obj_fruitWheatAle;

desc obj_cranberryWheat inherits obj_fruitWheatAle;

desc obj_blackBeer inherits obj_beer {
  prop_beer_medium_taste : taste;
};

desc act_sellbeer inherits action {
  obj_beer : BEER_SOLD;
};

desc act_recommendbeer inherits action {
  obj_beer : BEER_RECOMMENDED;
};

goal SellBeer {
  description:
   [ act_sellbeer
     BEER_SOLD [ obj_beer
                 name [ string ]
               ]
   ]
  min:
    1
  max:
    1
  cancel:
    [ bot ]
  binding:
   BeerPackage::sellBeer individual : [BEER_SOLD];
```

```
};

goal RecommendBeer {
  description:
   [ act_recommendbeer
      BEER_RECOMMENDED [ obj_beer ]
   ]
  min:
    1
  max:
    1
  cancel:
   [ bot ]
  binding:
   BeerPackage::recommendBeer individual : [BEER_RECOMMENDED];
};



rules english {
<prop_beer_sweet:A:_>          = 'sweet';
<prop_beer_very_sweet:A:_>     = 'very sweet';
<prop_beer_bitter:A:_>         = 'bitter'
                               : 'dry'
                               : 'tart';
<prop_beer_very_bitter:A:_>    = 'very bitter'
                               : 'very dry'
                               : 'very tart';
<prop_beer_medium_taste:A:_>   = 'medium tasting'
                               : 'medium'
                               : 'medium bitter';

<prop_beer_light:A:_>          = 'light'
                               : 'light bodied'
                               : 'refreshing';
<prop_beer_very_light:A:_>     = 'very light'
                               : 'very light bodied'
                               : 'very refreshing';
<prop_beer_full:A:_>           = 'full'
                               : 'heavy'
                               : 'filling'
                               : 'full bodied'
                               : 'rich';
<prop_beer_very_full:A:_>      = 'very full'
                               : 'very heavy'
                               : 'very filling'
                               : 'very full bodied'
                               : 'very rich';
<prop_beer_medium_body:A:_>    = 'medium bodied'
                               : 'medium';

<BEERWORD>                     = 'beer';
<FRUIT>                       = 'fruit' { fruit true };
<obj_beer:N:_>                 = 'beer';
<obj_ale:N:_>                  = 'ale';
<obj_americanAle:N:_>          = 'american ale';
<obj_stout:N:_>                = 'stout';
<obj_dryStout:N:_>             = 'dry stout';
<obj_oatmealStout:N:_>         = 'oatmeal stout';
<obj_imperialStout:N:_>        = 'imperial stout';
<obj_sweetStout:N:_>           = 'sweet stout';
<obj_paleAle:N:_>              = 'pale ale';
<obj_brownAle:N:_>             = 'brown ale';
<obj_oldAle:N:_>               = 'old ale';
<obj_barleywine:N:_>           = 'barleywine';
<obj_scotchAle:N:_>            = 'scotch ale';
<obj_irishAle:N:_>             = 'irish ale';
<obj_belgianAle:N:_>           = 'belgian ale';
<obj_goldenAle:N:_>            = 'golden ale';
<obj_trappistAle:N:_>          = 'trappist ale';
<obj_altbier:N:_>              = 'altbier';
<obj_indiaPaleAle:N:_>         = 'india pale ale';
<obj_porter:N:_>               = 'porter';
<obj_lager:N:_>                = 'lager';
<obj_marzen:N:_>               = 'marzen';
<obj_pilsner:N:_>              = 'pilsner';
<obj_lightAmericanPilsner:N:_> = 'light american pilsner';
<obj_premium:N:_>               = 'premium'
                                : 'premium beer';
<obj_subpremium:N:_>            = 'subpremium'
                                : 'cheap beer';
<obj_superpremium:N:_>          = 'superpremium';
<obj_dortmunderExport:N:_>      = 'dortmunder export';
<obj_bock:N:_>                  = 'bock';
<obj_dopplebock:N:_>            = 'dopplebock'
                                : 'double bock';
<obj_steamBeer:N:_>             = 'steam beer';
<obj_wheatAle:N:_>              = 'wheat beer'
                                : 'wheat ale'
                                : 'weizen';
```

```
<obj_weisse:N:_>               = 'weisse';
<obj_berlinerWeisse:N:_>       = 'berliner weisse';
<obj_lambic:N:_>               = 'lambic';
<obj_fruitLambic:N:_>          = 'fruit beer';
<obj_raspberryLambic:N:_>      = 'raspberry lambic';
<obj_peachLambic:N:_>          = 'peach lambic';
<obj_cherryLambic:N:_>         = 'cherry lambic';
<obj_fruitWheatAle:N:_>        = 'fruit ale';
<obj_cherryWheat:N:_>          = 'cherry wheat';
<obj_cranberryWheat:N:_>       = 'cranberry lambic'
                               : 'cranberry wheat';
<obj_blackBeer:N:_>            = 'black beer';

<BEERBYORIGIN>                 = import dao apps/beer/databases/beersDB.mdb beers origin {origin import};

<obj_beer:N:_>                 = import dao apps/beer/databases/beersDB.mdb beers name {name import}
                               : <prp_nationality:A:_> { origin prp_nationality } <BEERWORD>
                               : <prp_nationality:A:_>* { origin prp_nationality }
                                    <prop_beer_body:A:_> {body prop_beer_body} <BEERWORD>
                               : <prp_nationality:A:_>* { origin prp_nationality }
                                    <prop_beer_body:A:_>* {body prop_beer_body}
                                    <prop_beer_taste:A:_> {taste prop_beer_taste} <BEERWORD>
                               : <prp_nationality:A:_>* { origin prp_nationality }
                                    <prop_beer_taste:A:_> {taste prop_beer_taste}
                                    <prop_beer_body:A:_>* {body prop_beer_body} <BEERWORD>
                               : <FRUIT>  <BEERWORD>;

public <act_recommendbeer:V:_>        = <infoRequest> <obj_beer:N:_> {BEER_RECOMMENDED obj_beer};
public <act_sellbeer:V:_>             = <purchaseRequest> <obj_beer:N:_> {BEER_SOLD obj_beer};

<infoRequest>                  = 'do you have a'
                               : 'can you recommend a'
                               : 'im looking for a';

<purchaseRequest>              = 'ill take a'
                               : 'give me a'
                               : 'id like a';

};

database beersDB obj_beer dao apps/beer/databases/beersDB.mdb {
  dbtable beers obj_beer {
    dbfield type   = [];
    dbfield name   = [name];
    dbfield origin = [origin];
  };
};

templates english {
statement {
        state:(finalized = RecommendBeer) ->
        text:  "I recommend the following beers
        ##objs^first@{[BEER_RECOMMENDED|name] from [BEER_RECOMMENDED|origin]:<prp_nationality:N:_> }
          ##objs^middle@{ , [BEER_RECOMMENDED|name] from [BEER_RECOMMENDED|origin]:<prp_nationality:N:_> }
        ##objs^last@{ and [BEER_RECOMMENDED|name] from [BEER_RECOMMENDED|origin]:<prp_nationality:N:_> } ."
      };

      statement {
        state:(finalized = SellBeer) ->
        text:  "Enjoy your ##objs^first@[BEER_SOLD|name] ."
      };

     enumqst {
        path:(ambiguous = ##objs@[BEER_SOLD|origin]) ->
        text: "Would you like a beer from
          ##objs^first@[BEER_SOLD|origin]:<prp_nationality:N:_> ,
          ##objs^middle@[BEER_SOLD|origin]:<prp_nationality:N:_> ,
          or
          ##objs^last@[BEER_SOLD|origin]:<prp_nationality:N:_> . "

         options: " ##objs^all@[BEER_SOLD|origin]:<prp_nationality:N:_> "
         commands: 'help' 'start over'
         location: [BEER_SOLD] <obj_beer:N:_>
     };

      enumqst {
        path:(ambiguous = ##objs@[BEER_SOLD|name]) ->
        text: "I have the following beers
          ##objs^first@[BEER_SOLD|name]
          ##objs^first@{ from [BEER_SOLD|origin]:<prp_nationality:N:_> }
          ##objs^middle@[BEER_SOLD|name]
          ##objs^middle@{ from [BEER_SOLD|origin]:<prp_nationality:N:_> }
          ##objs^last@{ and [BEER_SOLD|name]}
          ##objs^last@{ from [BEER_SOLD|origin]:<prp_nationality:N:_> } .
          Which would you like ?"
         options: " ##objs^all@[BEER_SOLD|name] "
         commands: 'help' 'start over'
         location: [BEER_SOLD] <obj_beer:N:_>
     };
```

```
      enumqst {
          path:(ambiguous = ##objs@[BEER_SOLD]) ->
          text: "I have the following types of beers .
          ##objs^first@[BEER_SOLD]:<_:N:_> ,
          ##objs^middle@[BEER_SOLD]:<_:N:_> ,
          and
          ##objs^last@[BEER_SOLD]:<_:N:_> "
          options: " ##objs^all@[BEER_SOLD]:<_:N:_> "
          commands: 'help' 'start over'
          location: [BEER_SOLD] <obj_beer:N:_>
      };

      enumqst {
         path:(ambiguous = ##objs@[BEER_SOLD|taste]) ->
         text: "Would you prefer a beer that tastes
           ##objs^first@[BEER_SOLD|taste]:<_:A:_> ,
           ##objs^middle@[BEER_SOLD|taste]:<_:A:_> ,
           or
           ##objs^last@[BEER_SOLD|taste]:<_:A:_> ? "
         options: " ##objs^all@[BEER_SOLD|taste]:<_:A:_> "
         commands: 'help' 'start over'
         location: [BEER_SOLD] <obj_beer:N:_>
      };

      enumqst {
         path:(ambiguous = ##objs@[BEER_SOLD|body]) ->
         text: "Would you prefer a
           ##objs^first@[BEER_SOLD|body]:<_:A:_> ,
           ##objs^middle@[BEER_SOLD|body]:<_:A:_> ,
           or
           ##objs^last@[BEER_SOLD|body]:<_:A:_> ? "
         options: " ##objs^all@[BEER_SOLD|body]:<_:A:_> "
         commands: 'help' 'start over'
         location: [BEER_SOLD] <obj_beer:N:_>
      };

  };

}
```

A.2 Anrufvermittlersystem

Der folgende Ausdruck zeigt die Spezifikation des Anrufvermittlersystems.

```
package islDirectory;

server: ExamplePackage localhost 5454;

import generic.dlm;

module islDirectory ExamplePackage {

  desc obj_employee inherits generic:object {
    base:string :   FirstName;
    base:string :   LastName;
    base:string :   Title;
    base:string :   DepartmentName;
    base:string :   EmailName;
    base:string :   HomePhone;
    base:string :   WorkPhone;
    base:string :   OfficeLocation;
  };


  desc employeeObject inherits generic:action {
    obj_employee : EMPLOYEE;
  };



  desc act_lookup inherits employeeObject {
  };

  desc act_call inherits employeeObject  {
  };


  goal lookup {
    description:
      [ act_lookup
        EMPLOYEE [ obj_employee
                 FirstName [ base:string ]
                 LastName [ base:string ]
               ]
      ]
    min:
      1
    max:
      1
    binding:
      ardpkg://localhost:5454/lookup    individual: [EMPLOYEE|FirstName] , [EMPLOYEE|LastName];
  };

  goal call {
    description:
      [ act_call
        EMPLOYEE [ obj_employee
                 WorkPhone [ base:string ]
               ]
      ]
    min:
      1
    max:
      1
    binding:
      ardpkg://localhost:5454/call    individual: [EMPLOYEE|WorkPhone] ;
  };


  rules english {
     <act_call,V,_>    lmclass  = 'call';
     <act_lookup,V,_>  lmclass  = 'look' 'up'
                               : 'lookup'
                               : 'find' ;

    public <generic:action,VP,_>= <act_call,V,_> <obj_employee,NP,_>* { EMPLOYEE obj_employee }
: <act_lookup,V,_> <obj_employee,NP,_>* { EMPLOYEE obj_employee } ;

    <obj_employee,NP,_>= <FirstNameGRS>
: <LastNameGRS>
: <FirstNameGRS> <LastNameGRS>;

    <FirstNameGRS> lmclass = import daoitf apps/islDirectory/databases/directory.mdb Employees FirstName
{ FirstName import };
    <LastNameGRS>  lmclass = import daoitf apps/islDirectory/databases/directory.mdb Employees LastName
{ LastName import };

  };
```

```
database Employees obj_employee daoitf apps/islDirectory/databases/directory.mdb {

  dbtable Employees obj_employee {

    dbfield FirstName    = [FirstName];
    dbfield LastName     = [LastName];
    dbfield Title    = [Title];
    dbfield DepartmentName  = [DepartmentName];
    dbfield EmailName     = [EmailName];
    dbfield HomePhone     = [HomePhone];
    dbfield WorkPhone     = [WorkPhone];
    dbfield OfficeLocation  = [OfficeLocation];

  };
};


templates english {

  enumqst {
    path:(ambiguous = ##goals@[]) ->
      text:     " Would you like to call or lookup
                      ##sem@{[EMPLOYEE|FirstName] [EMPLOYEE|LastName]}?"
      commands: 'undo' 'repeat' 'start over' 'call' 'lookup'
  };


  enumqst {
    path:(ambiguous = ##objs@[EMPLOYEE|FirstName]),
    path:(unique    = ##objs@[EMPLOYEE|LastName])  ->
      text:     "There are ##objs^num employees with the last name
                 ##sem@[EMPLOYEE|LastName]
                 ##objs^first@[EMPLOYEE|FirstName],
                 ##objs^middle@[EMPLOYEE|FirstName] and
                 ##objs^last@[EMPLOYEE|FirstName]. What is the first name of the person you would like to call? "
      options:  "  ##objs^first@[EMPLOYEE|FirstName]
                 ##objs^middle@[EMPLOYEE|FirstName]
                 ##objs^last@[EMPLOYEE|FirstName]"
      commands: 'undo' 'repeat' 'start over'
      location: [EMPLOYEE] <obj_employee,NP,_>
  };

  enumqst {
    path:(ambiguous = ##objs@[EMPLOYEE|LastName]),
    path:(unique    = ##objs@[EMPLOYEE|FirstName])  ->
      text:     "There are ##objs^num
                 ##sem@[EMPLOYEE|FirstName] called
                 ##objs^first@[EMPLOYEE|LastName],
                 ##objs^middle@[EMPLOYEE|LastName] and
                 ##objs^last@[EMPLOYEE|LastName]. What is the last name of the person you would like to call? "
      options:  "  ##objs^first@[EMPLOYEE|LastName]
                 ##objs^middle@[EMPLOYEE|LastName]
                 ##objs^last@[EMPLOYEE|LastName]"
      commands: 'undo' 'repeat' 'start over'
      location: [EMPLOYEE] <obj_employee,NP,_>
  };



  infoqst {
   state:(determined = lookup),
   path:(undefined = ##sem@[EMPLOYEE|FirstName]) ->
      text:       "What is the first name of the person you would like to lookup?"
      options:  " ##db^apps/isldirectory/databases/directory@mdb@Employees@FirstName "
      commands: 'undo' 'repeat' 'start over' 'i dont know it'
      location:   [EMPLOYEE] <obj_employee,NP,_>
  };
  infoqst {
   state:(determined = lookup),
   path:(undefined = ##sem@[EMPLOYEE|LastName]) ->
      text:       "What is the last name of the person you would like to lookup?"
      options:  " ##db^apps/isldirectory/databases/directory@mdb@Employees@LastName "
      commands: 'undo' 'repeat' 'start over' 'i dont know it'
      location:   [EMPLOYEE] <obj_employee,NP,_>
  };
  infoqst {
   state:(determined = call),
   path:(undefined = ##sem@[EMPLOYEE|FirstName]) ->
      text:       "What is the first name of the person you would like to call?"
      options:  " ##db^apps/isldirectory/databases/directory@mdb@Employees@FirstName "
      commands: 'undo' 'repeat' 'start over' 'i dont know it'
      location:   [EMPLOYEE] <obj_employee,NP,_>
  };
  infoqst {
   state:(determined = call),
   path:(undefined = ##sem@[EMPLOYEE|LastName]) ->
      text:       "What is the last name of the person you would like to call?"
      options:  " ##db^apps/isldirectory/databases/directory@mdb@Employees@LastName "
      commands: 'undo' 'repeat' 'start over' 'i dont know it'
      location:   [EMPLOYEE] <obj_employee,NP,_>
```

```
    };
    statement {
      state:(finalized = call ) ->
        text:     "Calling ##objs^first@[EMPLOYEE|FirstName]
                         ##objs^first@[EMPLOYEE|LastName].  Please wait."
    };

    statement {
      state:(finalized = lookup ) ->
        text:     "I looked up ##objs^first@[EMPLOYEE|FirstName]
                           ##objs^first@[EMPLOYEE|LastName].
                           The office number is ##objs^first@[EMPLOYEE|OfficeLocation] . "
    };
  };
}
```

A.3 Eintrittskarten

Der folgende Ausdruck zeigt die Spezifikation des Eintrittskartenverkaufssystems.

```
package kennywood;

server: ExamplePackage localhost 5454;

import generic.dlm;

module kennywood ExamplePackage {

desc obj_ticket inherits object {
string : TICKETTYPE;
string : VISITTIME;
int    : PRICE;
};

desc act_purchasing inherits action {
obj_ticket : TICKET;
};

desc act_addtocart inherits act_purchasing;
desc act_removefromcart inherits act_purchasing;
desc act_totalcart inherits action;

  goal addtocart {
    description:
      [ act_addtocart
        TICKET [ obj_ticket
                 QUANTIFIER [ gq
                              NUMBER    [ int    ]
                            ]
                 TICKETTYPE [ string ]
                 VISITTIME  [ string ]
                 PRICE      [ int    ]
               ]
      ]
    min:
      1
    max:
      1
    cancel:
      [ bot ]
    binding:
      KennywoodPackage::addtocart  individual : [TICKET|QUANTIFIER|NUMBER], [TICKET|TICKETTYPE], [TICKET|VISITTIME], [TICKET|PRICE];
  };

  goal removefromcart {
    description:
      [ act_removefromcart
        TICKET [ obj_ticket
                 QUANTIFIER [ gq
                              NUMBER    [ int    ]
                            ]
                 TICKETTYPE [ string ]
                 VISITTIME  [ string ]
                 PRICE      [ int    ]
               ]
      ]
    min:
      1
    max:
      1
    cancel:
      [ bot ]
    binding:
      KennywoodPackage::removefromcart  individual : [TICKET|QUANTIFIER|NUMBER], [TICKET|TICKETTYPE], [TICKET|VISITTIME], [TICKET|PRICE];
  };

  goal totalcart {
    description:
      [ act_totalcart ]
    min:
      1
    max:
      1
    cancel:
      [ bot ]
    binding:
      KennywoodPackage::totalcart individual;
  };

  rules english {

public <act_addtocart:VP:_> implements ActionVP :
act_addtocart = 0 { "" act_addtocart },
obj_ticket = 1 { TICKET obj_ticket };
```

```
<obj_ticket:NP:_> implements ObjectNP :
obj_ticket = 0;

  <act_addtocart:V:_> = <Want>* <Buy>
: <Want>
: <CanI> <Buy>
: <CanYou>* <GiveMe>
: <WillYou> <GiveMe>;

public <act_removefromcart:VP:_> implements ActionVP :
        act_removefromcart = 0  { "" act_removefromcart },
        obj_ticket  = 1         { TICKET obj_ticket };

  <act_removefromcart:V:_>  = <Want>* <Cancel>
: 'do' 'not' <Want>
: <CanI> <Cancel>
: <CanYou> <Cancel>
: <WillYou> <Cancel>;

  public <act_totalcart:VP:_>= 'what' 'is' <Total>
: 'how' 'much' 'is' <Total>;


<Want>lmclass = 'want' 'to'*
: 'would' 'like' 'to'*
: 'need';

  <CanI> lmclass = 'can' 'i'
: 'could' 'i';

  <CanYou>lmclass = 'can' 'you'
: 'could' 'you';

  <WillYou>lmclass = 'will' 'you'
: 'would' 'you';

  <Buy>lmclass = 'buy'
   : 'get'
: 'add'
: 'order'
: 'place' 'an' 'order';


  <GiveMe>lmclass = 'give' 'me'
: 'get' 'me'
: 'sell' 'me';

  <Cancel> lmclass = 'cancel'
: 'give' 'back'
: 'take' 'away'
: 'subtract'
: 'cancel' 'an' 'order';

  <Total>lmclass = 'the' 'total' <Cost>*
: 'my' 'total' <Cost>*;

  <Cost>lmclass = 'cost'
: 'price';

<obj_ticket:N:_>= <VisitTime>* <TicketType>* <Ticket>
: <TicketType>* <Ticket> <For>* <VisitTime>
: <TicketType>           <For>* <VisitTime>*
: <VisitTime>;

  <VisitTime>lmclass = 'weekday' { VISITTIME "weekday" }
: 'weekend' { VISITTIME "weekend" };

  <TicketType>lmclass = 'general admission' { TICKETTYPE "general admission" }
: 'ride all day' { TICKETTYPE "ride all day" };

  <Ticket>lmclass = 'ticket'
: 'tickets';

  <For>= 'for' 'a'
: 'for' 'the'
: 'for' 'this';

  };

  database kennywood obj_ticket dao apps/kennywood/databases/kennywood.mdb {

    dbtable Products obj_ticket {

      dbfield VisitTime   = [VISITTIME];
      dbfield TicketType  = [TICKETTYPE];
      dbfield Price       = [PRICE];

    };
  };
```

```
  templates english {

    statement {
      state:(finalized = addtocart),
      equals:(1 = ##objs@[TICKET|QUANTIFIER|NUMBER])  ->
        text:     "You have ordered
                  ##objs^first@[TICKET|QUANTIFIER|NUMBER]
              ##objs^first@[TICKET|TICKETTYPE]
                  ticket for the
                  ##objs^first@[TICKET|VISITTIME]."
    };

    statement {
      state:(finalized = addtocart) ->
        text:     "You have ordered ##objs^first@[TICKET|QUANTIFIER|NUMBER]
               ##objs^first@[TICKET|TICKETTYPE]
                   tickets for the
                   ##objs^first@[TICKET|VISITTIME]."
    };

    statement {
      state:(finalized = removefromcart),
      equals:(1 = ##objs@[TICKET|QUANTIFIER|NUMBER])  ->
        text:     "I have cancelled your order for ##objs^first@[TICKET|QUANTIFIER|NUMBER]
               ##objs^first@[TICKET|TICKETTYPE]
                   ticket for the
                   ##objs^first@[TICKET|VISITTIME]."
    };

    statement {
      state:(finalized = removefromcart) ->
        text:     "I have cancelled your order for ##objs^first@[TICKET|QUANTIFIER|NUMBER]
               ##objs^first@[TICKET|TICKETTYPE]
                   tickets for the
                   ##objs^first@[TICKET|VISITTIME]."
    };

    enumqst {
      path:(ambiguous = ##goals@[]) ->
        text:   " Would like to place an order or to cancel an order for
                  ##sem@{[TICKET|TICKETTYPE]} tickets?"
        commands: 'place an order' 'cancel an order' 'undo' 'repeat' 'start over'
        location: [] <act_addtocart:VP:_> <act_removefromcart:VP:_>
    };

    enumqst {
      path:(ambiguous   = ##objs@[TICKET|VISITTIME]) ->
        text:   "I need a bit more information. Would that be
                ##sem@{[TICKET|QUANTIFIER|NUMBER]}
                ##sem@{[TICKET|TICKETTYPE]}
                tickets for a
                ##objs^first@[TICKET|VISITTIME]
                ##objs^middle@[TICKET|VISITTIME]
                or a
                ##objs^last@[TICKET|VISITTIME]?"
      options:  "  ##objs^first@[TICKET|VISITTIME]
              ##objs^middle@[TICKET|VISITTIME]
              ##objs^last@[TICKET|VISITTIME]"
        commands: 'undo' 'repeat' 'start over'
      location:  [TICKET] <obj_ticket:N:_>
    };

    enumqst {
      path:(ambiguous   = ##objs@[TICKET|TICKETTYPE]) ->
        text:   "I need a bit more information. Would that be
     ##sem@{[TICKET|QUANTIFIER|NUMBER]}
     ##objs^first@[TICKET|TICKETTYPE]
     ##objs^middle@[TICKET|TICKETTYPE]
   tickets or
     ##sem@{[TICKET|QUANTIFIER|NUMBER]}
     ##objs^last@[TICKET|TICKETTYPE]
   tickets for a
     ##sem@{[TICKET|VISITTIME]}?"
     options:  "  ##objs^first@[TICKET|TICKETTYPE]
     ##objs^middle@[TICKET|TICKETTYPE]
     ##objs^last@[TICKET|TICKETTYPE]"
        commands: 'undo' 'repeat' 'start over'
     location:  [TICKET] <obj_ticket:N:_>
    };

    infoqst {
      path:(undefined   = ##sem@[TICKET|QUANTIFIER|NUMBER]) ->
      text:     "How many ##sem@{[TICKET|TICKETTYPE]} tickets ##sem@{ for a [TICKET|VISITTIME]}?"
        commands: 'undo' 'repeat' 'start over' 'one' 'two' 'three' 'four' 'five'
        location:  [TICKET|QUANTIFIER] <gq_cardinal:_:_>
    };
  };

}
```

A.4 Roulette

Der folgende Ausdruck zeigt die Spezifikation des Roulettesystems.

```
package RoulettePackage;

server: RoulettePackage localhost 5454;

import generic.dlm;

module roulette ExamplePackage {
  desc obj_player inherits object {
    int: ID;
    string: Vorname;
    string: Nachname;
    property: Partei;
  };

  desc act_bet inherits action {
      obj_player: Spieler;
      int:    Euro;
  };

  desc act_betOnColor inherits act_bet {
      string: Color;
  };

  desc act_betOnParity inherits act_bet {
      string: Parity;
  };

  desc act_betOnNumber inherits act_bet {
      int: Number;
  };

  desc obj_celebrity inherits object {
    string :   FIRST_NAME;
    string :   LAST_NAME;
    property:  CHARACTERISTIC;
  };



  desc act_playGame inherits action;

  desc prp_CDU inherits property;

  desc prp_CSU inherits property;

  desc prp_SPD inherits property;

  desc prp_FDP inherits property;

  desc prp_Gruene inherits property;

  goal betOnColor {
    description:
      [ act_betOnColor
          Spieler [ obj_player
                      ID [ int ]
                  ]
          Euro      [ int ]
          Color      [ string ]
      ]
    min:
        1
    max:
        1
    cancel:
        [ bot ]
    binding:
      RoulettePackage::betOnColor individual: [Spieler|ID], [Euro], [Color];
  };

  goal betOnParity {
    description:
    [ act_betOnParity
    Spieler [ obj_player
            ID [ int ]
          ]
    Euro      [ int ]
    Parity  [ string ]
    ]
    min:
    1
    max:
    1
    cancel:
    [ bot ]
```

```
  binding:
    RoulettePackage::betOnParity individual: [Spieler|ID], [Euro], [Parity];
};

goal betOnNumber {
  description:
  [ act_betOnNumber
  Spieler [ obj_player
          ID [ int ]
        ]
  Euro      [ int ]
  Number  [ int ]
  ]
  min:
  1
  max:
  1
  cancel:
  [ bot ]
  binding:
    RoulettePackage::betOnNumber individual: [Spieler|ID], [Euro], [Number];
};

goal playGame {
  description:
  [ act_playGame
  ]
  min:
  1
  max:
  1
  cancel:
  [ bot ]
  binding:
  RoulettePackage::playGame individual ;
};



rules generic {
  public <action:VP:Imp>  implements SpeechActVP  : obj_player = 1 { Spieler obj_player }, act_bet = 0 { "" act_bet };
  <obj_player:NP:_>       implements ObjectNP              : obj_player = 0;
};



rules german {

    public <act_bet:VP:_>        = <act_betOnColor:VP:_>
                                   : <act_betOnNumber:VP:_>
                                   : <act_betOnParity:VP:_>
                         : <act_betOnColor:VP:_>
                                   : <act_betOnNumber:VP:_>
                                   : <act_betOnParity:VP:_>;


    public <act_betOnColor:VP:_>  =   <act_bet:V:_>* <Money>* <Color> <obj_player:PP:_>* {Spieler obj_player}:
                                          <act_bet:V:_>* <obj_player:PP:_>* {Spieler obj_player} <Money>* <Color>:
                                          <act_bet:V:_>* <Money>* <obj_player:PP:_>* {Spieler obj_player} <Color>;

    public <act_betOnNumber:VP:_> =   <act_bet:V:_>* <Money>* <Number> <obj_player:PP:_>* {Spieler obj_player}:
                                        <act_bet:V:_>* <obj_player:PP:_>* {Spieler obj_player} <Money>* <Number>:
                                        <act_bet:V:_>* <Money>* <obj_player:PP:_>* {Spieler obj_player} <Number>;

    public <act_betOnParity:VP:_> = <act_bet:V:_>* <Money>* <Parity> <obj_player:PP:_>* {Spieler obj_player}:
                                      <act_bet:V:_>* <obj_player:PP:_>* {Spieler obj_player} <Money>* <Parity>:
                                      <act_bet:V:_>* <Money>* <obj_player:PP:_>* {Spieler obj_player} <Parity>;

    public <act_playGame:VP:_>    = 'spiele';

    public <act_bet:VP:_>           = <act_bet:V:_> <Money>* <obj_player:PP:_>* {Spieler obj_player}
                               : <act_bet:V:_> <obj_player:PP:_>* {Spieler obj_player} <Money>*;

    <act_bet:V:_>                 = 'setze';

    <Money> = '100 euro' {Euro 100};
    <Color> = 'auf rot' { Color "rot" }:
              'auf schwarz' { Color "schwarz" }:
               'auf'* 'eine'* 'farbe';


    <Parity> =  'auf gerade' { Parity "gerade" }:
                'auf ungerade' { Parity "ungerade" } :
                'auf'* 'eine'* 'Paritaet';



    <Number> =  'auf 1' { Number 1 }:
                   'auf'* 'eine'* 'Zahl';
```

```
    <obj_player:PP:_> = 'fuer' <obj_player:NP:_>;
    <obj_player:N:_>  = <Vorname> <Nachname>
                      : <Vorname>
                      : <Nachname>;

    <Vorname>     = import dao apps/roulette/databases/RoulettePlayers.MDB Personen Vorname {Vorname import };
    <Nachname>    = import dao apps/roulette/databases/RoulettePlayers.MDB Personen Nachname {Nachname import };
    <prp_CDU:N:_> = 'CDU'
                    : 'Christlich Demokratische Union';
    <prp_CSU:N:_> = 'CSU'
                    : 'Christlich Soziale Union';

    <prp_SPD:N:_> = 'SPD'
                    : 'Sozialdemokratische Partei Deutschlands';

    <prp_FDP:N:_> = 'FDP'
                    : 'Freiheitlich Demokratische Partei';

    <prp_Gruene:N:_>  = 'Gruene'
                        : 'Buendnis 90 die Gruenen';

};

  database RoulettePlayers obj_player dao apps/roulette/databases/RoulettePlayers.MDB {

      dbtable Personen obj_player {
        dbfield ID = [ID];
        dbfield Nachname  = [Nachname];
        dbfield Vorname   = [Vorname];
      };

};



templates german {

    statement {
      state:(finalized = playGame) ->
        text: "Rien ne va plus. "
    };

    statement {

      state:(finalized = betOnColor) ->
      text: "Fuer ##objs^first@[Spieler|Vorname]
                 ##objs^first@[Spieler|Nachname] wurden
                 ##objs^first@[Euro] Euro auf
                 ##objs^first@[Color] gesetzt."
    };

    statement {

      state:(finalized = betOnParity) ->
      text: "Fuer ##objs^first@[Spieler|Vorname]
            ##objs^first@[Spieler|Nachname] wurden
            ##objs^first@[Euro] Euro auf
            ##objs^first@[Parity] gesetzt."
    };

    statement {
      state:(finalized = betOnNumber) ->
      text: "Fuer ##objs^first@[Spieler|Vorname]
                 ##objs^first@[Spieler|Nachname] wurden
                 ##objs^first@[Euro] Euro auf
                 ##objs^first@[Number] gesetzt."
    };

    enumqst {
      path:(ambiguous = ##goals@[]) ->
      text:     " Moechten Sie auf eine Farbe, eine Zahl oder auf eine Paritaet setzen?"
      location: [] <act_bet:VP:_>
    };

    enumqst {
      path:(ambiguous = ##objs@[Spieler|Vorname]),
      path:(unique    = ##objs@[Spieler|Nachname]) ->
        text: "Es gibt ##objs^num
               ##sem@[Spieler|Nachname] mit den Vornamen
               ##objs^first@[Spieler|Vorname],
               ##objs^middle@[Spieler|Vorname] und
               ##objs^last@[Spieler|Vorname].
               Welches ist der Vorname des Polikers, fuer den gesetzt werden soll? "

        options: " ##objs^first@[Spieler|Vorname]
                   ##objs^middle@[Spieler|Vorname]
                   ##objs^last@[Spieler|Vorname]"
        commands: 'help' 'repeat' 'start over' 'tell me more'
```

```
          location: [Spieler] <obj_player:NP:_>

       };

      enumqst {
        path:(ambiguous = ##objs@[Spieler|Nachname]),
        path:(unique    = ##objs@[Spieler|Vorname]) ->
          text: "Es gibt ##objs^num
                   ##sem@[Spieler|Vorname] mit den Nachnamen
                   ##objs^first@[Spieler|Nachname],
                   ##objs^middle@[Spieler|Nachname] und
                   ##objs^last@[Spieler|Nachname].
                   Welches ist der Nachname des Polikers, fuer den gesetzt werden soll? "
          options: " ##objs^first@[Spieler|Nachname]
                     ##objs^middle@[Spieler|Nachname]
                     ##objs^last@[Spieler|Nachname]"

          commands: 'help' 'repeat' 'start over' 'tell me more'

          location: [Spieler] <obj_player:NP:_>

      };


      infoqst {
      path:(undefined = ##sem@[Euro]) ->
          text:       "Wieviel soll fuer den Politiker gesetzt werden?"
          location:   [] <Money>
      };
      infoqst {
        state:(determined       = betOnParity),
        path:(undefined         = ##sem@[Parity]) ->
          text:       "Moechten Sie auf gerade oder ungerade setzen?"
          location:   [] <Parity>
      };

      infoqst {
        state:(determined       = betOnColor),
        path:(undefined         = ##sem@[Color]) ->
          text:       "Moechten Sie auf rot oder schwarz setzen?"
          location:   [] <Color>
      };

      infoqst {
        state:(determined       = betOnNumber),
        path:(undefined         = ##sem@[Number]) ->
          text:       "Auf welche Zahl moechten Sie setzen?"
          location:   [] <Number>
      };

      infoqst {
        path:(undefined = ##sem@[Spieler|ID]) ->
          text:       "Welches ist der Nachname des Polikers, fuer den gesetzt werden soll?"
          location:   [Spieler] <obj_player:NP:_>
      };

        infoqst {
        path:(undefined = ##sem@[Spieler|ID]) ->
              text:       "Welches ist der Vorname des Polikers, fuer den gesetzt werden soll?"
              location:   [Spieler] <obj_player:NP:_>
      };
  };
}
}
```

A.5 CD-Spieler

Der folgende Ausdruck zeigt die Spezifikation des CD-Spielersystems.

```
package carstereopackage;

server: CarstereopackagePackage localhost 5454;

import generic.dlm;

module carstereopackage  CarstereopackagePackage {

  desc prp_playerstate inherits property;
  desc prp_mute inherits prp_playerstate;
  desc prp_play inherits prp_playerstate;
  desc prp_stop inherits prp_playerstate;
  desc prp_pause inherits prp_playerstate;

  desc obj_switchable inherits object {
    bool : ON;
    };

  desc obj_varies inherits object {
    int : PERCENT;
  };

  desc obj_player inherits obj_switchable {
    prp_playerstate : PLAYERSTATE;
    obj_varies      : VOLUME;
    obj_varies      : FADER;
    };

  desc obj_cdplayer inherits obj_player;

  desc obj_playable inherits object {
    string : FILENAME;
    };
  desc obj_song inherits obj_playable {
        string                  : ARTIST;
        string                  : GENRE;
        string                  : NAME;
    };
  desc act_displayprg inherits action;
  desc act_changevolume inherits action {
    int     : CHANGEVOLUME;
  };

  desc act_switchonoff inherits action {
    obj_switchable  : ARG;
  };


  desc act_play inherits action {
    obj_song : SONG;
  };
  desc act_stopsong inherits action {
      obj_song : SONG;
  };
  desc act_nexttrack inherits action;
  desc act_prevtrack inherits action;
  desc act_switchoff inherits act_switchonoff;
  desc act_switchon inherits act_switchonoff;
  desc act_gototrack inherits action {
      int : TRACKNUMBER;
  };
  desc act_balance inherits action {
   int : CHANGEBALANCE;
  };

  goal balance {
     description:
     [ act_balance
       CHANGEBALANCE [ int ]
     ]
     cancel:
     [ bot ]
     binding:
     CarstereopackagePackage::changeBal individual : [CHANGEBALANCE];

  };

  goal gotoTrack {
       description:
       [ act_gototrack
         TRACKNUMBER [ int ]
       ]
       cancel:
       [ bot ]
       binding:
```

```
     CarstereopackagePackage::gotoTrack individual : [TRACKNUMBER];

};


goal changeVolume {
  description:
    [ act_changevolume
      CHANGEVOLUME [ int ]
    ]
  cancel:
    [ bot ]
  binding:
   CarstereopackagePackage::changeVol individual : [CHANGEVOLUME];

};


goal NextTrack {
  description:
    [ act_nexttrack ]
  cancel:
    [ bot ]
  binding:
    CarstereopackagePackage::nextTrack individual ;

};

goal PrevTrack {
  description:
    [ act_prevtrack ]
  cancel:
    [ bot ]
  binding:
    CarstereopackagePackage::prevTrack individual ;

};

goal PlaySong {
  description:
      [ act_play
         SONG    [ obj_song
                   NAME           [ string ]
                   FILENAME       [ string ]
                   GENRE [ string ]
                 ]
      ]
  min:
      1
  max:
      1
  cancel:
      [ bot ]
  binding:
      CarstereopackagePackage::playSong individual  : [SONG|FILENAME] , [SONG|NAME], [SONG|GENRE];
};

rules generic {
  public <action:VP:Imp>  implements SpeechActVP  : act_play = 0 , obj_playable = 1 { ARG object };
<obj_playable:NP:_>        implements ObjectNP     : obj_playable = 0;
<obj_song:NP:_> implements ObjectNP : obj_song = 0;
};

rules english {

  public <act_play:VP:_>= <act_play:V:_> <obj_song:NP:_> { SONG obj_song };
  public <act_nexttrack:VP:_>lmclass = 'next' 'track' : 'go to the next track' ;
  public <act_prevtrack:VP:_>lmclass = 'previous' 'track' : 'go to the previous track' ;
  public <act_gototrack:VP:_> lmclass = 'go' 'to' 'track'  <TRACK>
   : 'goto' 'track' <TRACK>
   : 'track' <TRACK>;
  public <act_balance:VP:_>lmclass = 'left' {CHANGEBALANCE 128}
     : 'right' {CHANGEBALANCE 127}
     : 'center' {CHANGEBALANCE 0};


  public <act_changevolume:VP:_> lmclass = 'louder' { CHANGEVOLUME 10 }
        : 'a little louder' { CHANGEVOLUME 5 }
        : 'a little bit louder' { CHANGEVOLUME 5 }
        : 'too soft' { CHANGEVOLUME 10 }
        : 'raise the volume' { CHANGEVOLUME 10 }
        : 'softer' { CHANGEVOLUME -10 }
        : 'quieter' { CHANGEVOLUME -10 }
        :'too loud' { CHANGEVOLUME -10 }
        : 'a little softer' { CHANGEVOLUME -5 }
        : 'a little bit softer' { CHANGEVOLUME -5 }
        : 'lower the volume' { CHANGEVOLUME -10 }
        : 'a lot louder' { CHANGEVOLUME 20 }
        : 'a lot softer' { CHANGEVOLUME -20 };
```

```
    <act_play:V:_>    lmclass = 'play' : 'play for me';
    <act_switchon:V:_>lmclass = 'switch on';
    <act_switchoff:V:_>lmclass = 'switch off';

    <obj_song:N:_>    = <Name>
                      : <Artist>
                      : <Genre>;

    <TRACK> lmclass = 'one' { TRACKNUMBER 1 }
              : 'two' { TRACKNUMBER 2 }
              : 'three' { TRACKNUMBER 3 }
              : 'four' { TRACKNUMBER 4 }
              : 'five' { TRACKNUMBER 5 }
              : 'six' { TRACKNUMBER 6 };

    <Name>lmclass   = import dao apps/carstereopackage/databases/carstereopackage.mdb Music Name { NAME import };
    <Artist> lmclass = import dao apps/carstereopackage/databases/carstereopackage.mdb Music Artist { ARTIST import };
    <Genre>lmclass   = import dao apps/carstereopackage/databases/carstereopackage.mdb Music Genre { GENRE import };



  };
  database CarstereoDB obj_song dao apps/carstereopackage/databases/carstereopackage.mdb {
    dbtable Music obj_song {
      dbfield Name    = [NAME];
      dbfield Artist  = [ARTIST];
      dbfield Genre  = [GENRE];
      dbfield Filename  = [FILENAME];

      guard Name {
      [NAME] > string
       };
      guard Artist {
      [ARTIST] > string
       };
      guard Genre {
      [GENRE] > string
       };
      guard Filename {
      [FILENAME] > string
       };
    };
  };


templates english {
  enumqst ListBox {
    path:(ambiguous = ##objs@[SONG|NAME]) ->
      text: " ##objs^first@[SONG|NAME],
      ##objs^middle@[SONG|NAME], and
      ##objs^last@[SONG|NAME]. What song would you like to hear? "
      options:  " ##objs^first@{[SONG|NAME] by [SONG|ARTIST]}
            ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]}
            ##objs^last@{[SONG|NAME] by [SONG|ARTIST]} "
      location: [SONG] <obj_song:NP:_>
};
  enumqst ListBox  {
    path:(defined   = ##sem@[SONG|NAME]),
    path:(ambiguous = ##objs@[SONG|ARTIST]) ->
      text:     "There are ##objs^num songs called
                      ##sem@[SONG|NAME]. What is the name of the artist you would like to hear? "
      options:  " ##objs^first@[SONG|NAME]
                 ##objs^middle@[SONG|NAME]
                 ##objs^last@[SONG|NAME] "
  };

  infoqst ListBox  {
    path:(ambiguous = ##objs@[SONG|NAME])   ->
     text:     "There are ##objs^num matching songs. What is the name of the song you would like to hear?"
      options:  " ##objs^first@{[SONG|NAME] by [SONG|ARTIST]}
                 ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]}
                 ##objs^last@{[SONG|NAME] by [SONG|ARTIST]}"
  };



  illegaldesc ListBox {
    path:(defined   = ##sem@[SONG|NAME]),
    path:(defined   = ##sem@[SONG|ARTIST]),
    less:(4         = ##objs@[SONG|NAME]),
    less:(4         = ##objs@[SONG|ARTIST])      ->
        text:     " There is no song ##sem@[SONG|NAME] by
                    ##sem@[SONG|ARTIST], but the following songs exist
                    ##objs^first@{[SONG|NAME] by [SONG|ARTIST]},
                    ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]} and
                    ##objs^last@{[SONG|NAME] by [SONG|ARTIST]}. "

        options:  " ##objs^first@{[SONG|NAME] by [SONG|ARTIST]}
                    ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]}
                    ##objs^last@{[SONG|NAME] by [SONG|ARTIST]}"
```

```
};

  illegaldesc ListBox {
path:(defined   = ##sem@[SONG|NAME]),
    path:(defined   = ##sem@[SONG|ARTIST]),
    less:(4         = ##objs@[SONG|NAME]),
    more:(3         = ##objs@[SONG|ARTIST])     ->
      text:     " There is no song ##sem@[SONG|NAME] by
                  ##sem@[SONG|ARTIST], but the following songs exist:
                  ##objs^first@<[SONG|NAME]>,
                  ##objs^middle@<[SONG|NAME]> and
                  ##objs^last@<[SONG|NAME]>. "
      options:  " ##objs^first@{[SONG|NAME] by [SONG|ARTIST]}
                  ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]} and
                  ##objs^last@{[SONG|NAME] by [SONG|ARTIST]} "
};

  illegaldesc ListBox {
    path:(defined   = ##sem@[SONG|NAME]),
    path:(defined   = ##sem@[SONG|ARTIST]),
    more:(3         = ##objs@[SONG|NAME]),
    less:(4         = ##objs@[SONG|ARTIST])      ->
      text:     " There is no song ##sem@[SONG|NAME] by
                  ##sem@[SONG|ARTIST], but the following songs exist
                  ##objs^first@<[SONG|ARTIST]>,
                  ##objs^middle@<[SONG|ARTIST]> and
                  ##objs^last@<[SONG|ARTIST]>.  "
      options:  " ##objs^first@{[SONG|NAME] by [SONG|ARTIST]}
                  ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]}
                  ##objs^last@{[SONG|NAME] by [SONG|ARTIST]} "
};
  illegaldesc ListBox {
    path:(defined   = ##sem@[SONG|NAME]),
    path:(defined   = ##sem@[SONG|ARTIST]),
    more:(3         = ##objs@[SONG|NAME]),
    more:(3         = ##objs@[SONG|ARTIST])      ->
      text:     " There is no song ##sem@[SONG|NAME] by
                  ##sem@[SONG|ARTIST], but there are some other songs you might be interested in. "
      options:  " ##objs^first@{[SONG|NAME] by [SONG|ARTIST]}
                  ##objs^middle@{[SONG|NAME] by [SONG|ARTIST]}
                  ##objs^last@{[SONG|NAME] by [SONG|ARTIST]}"
};
};

}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[Abella and Gorin, 1999] A. Abella and A.L. Gorin. Construct Algebra: Analytical Dialog Management. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 191–199, 1999.

[Allen *et al.*, 1996] J. Allen, B.W. Miller, E.K. Ringer, and T. Sikorski. Robust understanding in a dialogue system. In *Proceedings of the Meeting of the Association of Computational Linguistics*, 1996.

[Andersson *et al.*, 2001] E.A. Andersson, S. Breitenbach, T. Burd, N. Chidambaram, P. Houle, D. Newsome, X. Tang, and X. Zhu. *Early Adopter VoiceXML.* Wrox Press, 2001.

[Araki *et al.*, 1999] M. Araki, K. Komatani, T. Hirata, and S. Doshita. A Dialogue Library for Task-Oriented Spoken Dialogue Systems. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 1999.

[Bilange *et al.*, 1990] E. Bilange, M. Guyomard, and J. Siroux. Separating Dialogue Knowledge and Task Knowledge from Oral Dialogue Management. In *COGNITIVA'90, Madrid*, 1990.

[Bilange, 1991] E. Bilange. A Task Independent Oral Dialogue Model. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics, Berlin*. European Chapter of the Association for Computational Linguistics, 1991.

[Bobrow and Group, 1977] D. Bobrow and The PARC Understander Group. Gus-1, a frame driven dialog system. *Artificial Intelligence*, 8:155–173, 1977.

[Bouma, 1990] G. Bouma. Defaults in unification grammar. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL-90), Pittsburgh*, pages 165–173, 1990.

[Bouma, 1992] G. Bouma. Feature structures and nonmonotonicity. *Computational Linguistics*, ??:??, 1992.

[Bretier and Sadek, 1996] P. Bretier and M.D. Sadek. A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. In *Proceedings of the International Workshop on Agent Theories, Architectures, and Languages*, pages 189–203, 1996.

[Buo, 1996] F.D Buo. Feaspar - a feature structure parser learning to parse spontaneous speech, 1996.

[Carpenter, 1992] B. Carpenter. *The Logic of Typed Feature Structures.* Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1992.

[Carpenter, 1993] B. Carpenter. Skeptical and credulous default unification with application to templates and inheritance. In T. Briscoe, A. Copestake, and V. de Paiva, editors, *Inheritance, Defaults and the Lexicon*, pages 13–37. Cambridge University Press, 1993.

[Chu-Carroll and Carpenter, 1998] J. Chu-Carroll and B. Carpenter. Dialogue Management in vector-Based Call Routing. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98), Montreal, Canada*, pages 256–262, 1998.

[Chu-Carroll and Carpenter, 1999] J. Chu-Carroll and B. Carpenter. Vector-Based Natural Language Call Routing. *Journal of Computational Linguistics*, 24:361–388, 1999.

[Cole, 1999] R. Cole. Tools for Research and Education in Speech Science. In *Proceedings of the International Conference of Phonetic Sciences, San Francisco, USA*, 1999.

[Denecke and Waibel, 1997] M. Denecke and A.H. Waibel. Dialogue Strategies Guiding Users to their Communicative Goals. In *Proceedings of Eurospeech, Rhodos, Greece*, 1997. Available at http://www.is.cs.cmu.edu.

[Dörre and Eisele, 1990] J. Dörre and A. Eisele. Feature logic with disjunctive unification. In *Proceedings of the 13th International Conference on Computational Linguistics, Helsinki*, pages 100–105, 1990.

[Eisele and Dörre, 1992] A. Eisele and J. Dörre. Disjunctive unification. In *??*, pages 100–105. 1992.

[Ferrieux and M.D.Sadek, 1994] A. Ferrieux and M.D.Sadek. An Efficient Data-Driven Model for Cooperative Spoken Dialogue. In *Proceedings of the International Conference on Spoken Language Processing, Yokohama, Japan*, pages 979 – 982, 1994.

[Flycht-Eriksson, 1999] A. Flycht-Eriksson. A Survey of Knowledge Sources in Dialogue Systems. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Stockholm, Sweden*, 1999.

[Gavalda, 2000] M. Gavalda. Miso: A Parser for Real World Spontaneous Speech. In *Proceedings of the 6th International Workshop on Parsing Technologies, Trento, Italy*, 2000.

[Gorin *et al.*, 1997] A.L. Gorin, G. Riccardi, and J.H. Wright. How May I Help You. *Speech Communications*, 23:113–127, 1997.

[Gorin, 1996] A.L. Gorin. Processing Semantic Information in Fluently Spoken Language. In *Proceedings of the International Conference on Spoken Language Processing, Philadelphia, USA*, pages 1001–1004, 1996.

[Grosz and Sidner, 1986] B. Grosz and C. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, pages 175–204, 1986.

[Horrocks *et al.*, 1999] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for description logics with functional restrictions, inverse and transitive roles, and role hierarchies. In *Proceedings of the 1999 Workshop Methods for Modalities (M4M-1)*, Amsterdam, 1999.

[Kölzer, 1999] A. Kölzer. Universal Dialogue Specification for Conversational Systems. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 1999.

[Lascarides and Copestake, 1996] A. Lascarides and A. Copestake. Order independent and persistent typed default unification. *Linguistics and Philosophy*, 19:1–89, 1996.

[Lascarides and Copestake, 1999] A. Lascarides and A. Copestake. Default representations in constraint-based frameworks. *Computational Linguistics*, ??:??, 1999.

[Lenat, 1995] D. B. Lenat. Cyc: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38, 1995.

[Levin and Pieraccini, 1999] E. Levin and R. Pieraccini. Spoken Language Dialogue: From Theory to Practice. In *Proceedings of the Workshop on Automatic Speech Recognition and Understanding*, 1999.

[Levin *et al.*, 1998] L. Levin, D. Gates A. Lavie, F. Pianesi, D. Wallace, T. Watanabe, and M. Woszczyna. A Modular Approach for Spoken Language Translation in Large Domains. In *Proceedings of the AMTA*, 1998. Available at http://www.is.cs.cmu.edu.

[Ludwig *et al.*, 1998] B. Ludwig, G. Görz, and H. Niemann. Combining expression and content in domains for dialog managers. In *Proceedings of DL 98, ITC-Irst Technical Report 9805-03, Trento*, 1998.

[Miyao, 1999] Y. Miyao. Packing of feature structures for efficient unification of disjunctive feature structures. In *Proceedings of the 38th Conference of the ACL*, pages 579–584, 1999.

[Myers, 1991] B.A. Myers. Separating application code from toolkits: Eliminating the spaghetti of call-backs. In *ACM Symposium on User Interface Software and Technology: UIST'91, Hilton Head, SC, USA*, pages 211–220, 1991.

[Nebel and Smolka, 1991] B. Nebel and G. Smolka. Attribute-description formalisms... and the rest of the world. In O. Herzog and C. Rollinger, editors, *Textunderstanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence*, pages 439–452. Springer Verlag, 1991.

[Papineni *et al.*, 1999] K.A. Papineni, S. Roukos, and R.T. Ward. Free-Flow Dialogue Management Using Forms. In *Proceedings of EUROSPEECH 99, Budapest, Ungarn*, 1999.

[Pollard and Sag, 1994] C. Pollard and I.A. Sag. *Head Driven Phrase Structure Grammar*. Center for the Study of Language and Information, Stanford, CA, USA, 1994.

[Rayner *et al.*, 2001] M. Rayner, I. Lewin, G. Gorrell, and J. Boye. Plug and Play speech understanding. In *Proceedings of the* SIGDIAL *Workkshop on Discourse and Dialogue*, 2001.

[Rounds and Zhang, 1991] W.C. Rounds and G.Q Zhang. Suggestions for a non-monotonic feature logic. *Journal???*, 1991.

[Roy *et al.*, 2000] J. Roy, J. Pineau, and S. Thrun. Spoken Dialog Management for Robots. In *Association for Computational Linguistics (ACL 2000), Hong Kong*, 2000.

[Rudnicky and Wu, 1999] A. Rudnicky and X. Wu. An agenda-based Dialog Management Architecture for Spoken Language Systems. In *Proceedings of the Workshop on Automatic Speech Recognition and Understanding*, 1999.

[Sadek and De Mori, 1998] D. Sadek and R. De Mori. Dialogue Systems. In R. De Mori, editor, *Spoken Dialogues with Computers*, pages 524–561, 1998.

[Sadek *et al.*, 1997] M.D. Sadek, P. Bretier, and E. Panaget. ARTIMIS: Natural Dialogue Meets Rational Agency. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1030–1035, 1997.

[Singh *et al.*, 2002 to appear] S. Singh, D. Litman, M. Kearns, and M Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFUN System. *Journal of Artificial Intelligence Research (*JAIR*)*, 2002, to appear.

[Smith, 1992] R.W. Smith. Integration of domain problem solving with natural language dialog: The missing axiom theory. In *Proceedings of Applications of AI X: Knowledge-Based Systems*, pages 270–278, 1992.

[Soltau and Waibel, 2000] H. Soltau and A. Waibel. Specialized acoustic models for hyperarticulated speech. In *Proceedings of ICASSP, Istambul, Turkey*, 2000.

[Sutton *et al.*, 1996] S. Sutton, D.G. Novick, R.A. Cole, and M. Fanty. Building 10,000 Spoken Dialogue Systems. In *Proceedings of the International Conference on Spoken Language Processing, Philadelphia, PA*, 1996.

[Taylor *et al.*, 1998] P. Taylor, A. Black, and R Caley. The Architecture of the Festival Speech Synthesis System. In *3rd ESCA Workshop on Speech Synthesis*, pages 147–151, 1998.

[Traum and Hinkelman, 1992] D. Traum and E.A. Hinkelman. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, 8:575–599, 1992.

[Traum, 1999] D. Traum. 20 questions on dialogue act taxonomies. In *Proceedings of Amstelogue 99, Amsterdam, The Netherlands*, 1999.

[Vogel and Cooper, 1995] C. Vogel and R. Cooper. Robust chart parsing with mildly inconsistent feature structures. In A. Schöter and C. Vogel, editors, *Edinburgh Working Papers in Cognitive Science, Vol. 10*. HCRC Edinburgh, 1995.

[Walker *et al.*, 1997] M.A. Walker, D.J. Litman, C.A. Kamm, and A. Abella. Paradise: A framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, pages 271–280, 1997.

[Walker *et al.*, 2001] M.A Walker, O. Rambow, and M Rogati. A trainable approach to sentence planning for spoken dialogue. *In Submission*, 2001.

[Ward, 1994] W. Ward. Extracting Information in Spontaneous Speech. In *Proceedings of the International Conference on Spoken Language Processing, Yokohama, Japan*, pages 83–87, 1994.

[Woszczyna *et al.*, 2000] M. Woszczyna, M. Broadhead, D. Gates, M. Gavalda, A. Lavie, L. Levin, and A. Waibel. Evaluation of a practical interlingua for task-dependent dialogue. In *Proceedings of the AMTA SIG-IL Third Workshop on Interlinguas and Interlingua Approaches, Seattle, Washington*, 2000. Available at http://www.is.cs.cmu.edu.

[Wright *et al.*, 1998] J.H. Wright, A.L. Gorin, and A. Abella. Spoken Language Understanding Within Dialogs Using a Graphical Model of Task Structure. In *Proceedings of the International Conference on Spoken Language Processing, Sydney, Australia*, 1998.